DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMPRESIÓN DE IMÁGENES USANDO DISPOSITIVOS REPROGRAMABLES

Norma Ximena Ríos Cotazo

UNIVERSIDAD DEL VALLE

PROGRAMA DE POSGRADOS EN INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN ELECTRÓNICA

GRUPO DE ARQUITECTURAS DIGITALES Y MICROELECTRÓNICA

SANTIAGO DE CALI, AGOSTO DE 2011

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMPRESIÓN DE IMÁGENES USANDO DISPOSITIVOS REPROGRAMABLES

Norma Ximena Ríos Cotazo

Trabajo de grado para optar por el titulo de Magíster en Ingeniería Énfasis Electrónica

Director

Ing. Álvaro Bernal Noreña Ph.D.

UNIVERSIDAD DEL VALLE
PROGRAMA DE POSGRADOS EN INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN ELECTRÓNICA
GRUPO DE ARQUITECTURAS DIGITALES Y MICROELECTRÓNICA
SANTIAGO DE CALI, AGOSTO DE 2011

UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

Programa de Posgrados en Ingeniería Eléctrica y Electrónica
Maestría en Ingeniería Énfasis Electrónica

Título: Diseño e Implementación de un Sistema de Compresión De Imágenes Usando Dispositivos Reprogramables.

Autor: Ing. Norma Ximena Rios Cotazo

Director: Ing. Álvaro Bernal Noreña Ph.D.

Grupo de Investigación: Arquitecturas Digitales y Microelectrónica

Área de Investigación: Diseño de Sistemas Digitales Usando Arquitecturas Reprogramables

Descriptores: Transformada Wavelet, Compresión de Imágenes, Algoritmo SPIHT, Dispositivos Reprogramables, Procesador Nios.

Nota de Aprobación

El trabajo de grado titulado "**Diseño e Implementación de un Sistema de Compresión De Imágenes Usando Dispositivos Reprogramables"**, presentado por la estudiante NORMA XIMENA RÍOS COTAZO, para optar al titulo de Magíster en Ingeniería Énfasis Electrónica, fue revisado por el jurado y calificado como:

_____

_____

Ing. Álvaro Bernal Noreña Ph.D.

Director

_____

Jurado

_____

Jurado

RESUMEN

En este documento se presenta el diseño e implementación de un sistema de compresión de imágenes, el sistema utiliza un esquema de compresión por transformadas donde se usa la transformada wavelet para llevar la señal a un domino mas adecuado para su compresión, se inicia con un planteamiento teórico acerca de como representar una señal por medio de las funciones wavelet, seguido de una revisión del estado del arte de las bases mas utilizadas para el procesamiento de señales y la selección de la base wavelet que se utiliza en el sistema, posteriormente se realiza una revisión del estado del arte de las arquitecturas propuestas para la implementación de la transformada wavelet y se presenta el diseño e implementación de una arquitectura paralela por nivel, de igual forma se despliega una revisión de algunas de las técnicas de compresión de imágenes más representativas y se propone e implementa una modificación del algoritmo de compresión SPIHT (Set Partitioning in Hierarchical Trees). El sistema se implemento en la FPGA Ciclone II EP2C35F672C6 de Altera utilizando un diseño soportado en el sistema Nios II.

Palabras clave

Transformada Wavelet, Compresión de Imágenes, Algoritmo SPIHT, Dispositivos Reprogramables, Procesador Nios.

…Dedicatoria

... A Dios, por darme la fuerza e iluminar mi camino...

...A mi madre Norma, por su amor incondicional, por su invaluable apoyo, por todo...

...A mi hermano Oscar, a mi esposo Orlando, a mi Familia, por su amor y comprensión...

...A mis amigos, por su amistad y confianza...

Norma Ximena Rios

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

Al director del trabajo de grado: Ingeniero Álvaro Bernal, por su valiosa colaboración, su paciencia, guía y apoyo.

A mis amigos de maestría: Bayron, Jorge T., Juliana, Breyner, Fabio, Jorge E., por sus valiosos aportes, por su amistad y confianza.

A todos los profesores y personal del Posgrado y a la Universidad del Valle, por todo lo que aprendí y porque gracias a ellos hoy he llegado hasta aquí.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

Capítulo 1.

Introducción

En la actualidad son innumerables las aplicaciones térmicas, topográficas, médicas, de procesos industriales y multimedia, entre otras, que requieren procesar almacenar y transmitir imágenes digitales, este creciente uso de las imágenes como herramienta de análisis ha promovido el desarrollo de técnicas para adecuar sus características a los requerimientos del sistema del que hace parte. Una característica fundamental desde el punto de vista de la capacidad del sistema es el volumen de la información en los archivos de imágenes digitales que puede ser alto y ocupar mucho espacio, lo que genera la necesidad de usar técnicas para la compactación de la información con el fin de facilitar su transmisión y su almacenamiento; las técnicas de compresión intentan a través de algoritmos matemáticos reducir de diversos modos el tamaño del archivo que contiene la imagen, un buen algoritmo de compresión tiene dos metas[1]: proveer un método eficiente de representación de la imagen y descartar aquellas características a las que los sentidos humanos son menos sensibles o aquella información de sucesos que son muy poco probables.

Este trabajo de grado ha sido planteado en el marco del proyecto de investigación "Desarrollo e Implementación de un Prototipo de Acompañante Móvil Digital" [48] desarrollado por el grupo de Arquitecturas Digitales y Microelectrónica de la Universidad del Valle en asociación con el grupo de Microelectrónica y Control de la Universidad de Antioquia. El proyecto pretende el desarrollo de una arquitectura abierta que sea eficiente en funciones multimedia, para lo cual se evaluará el uso de un sistema que integre

dispositivos reconfigurables con un procesador de propósito general. Dentro del marco de este proyecto surge la necesidad de desarrollar una arquitectura para la compresión y descompresión de imágenes adecuada para ser implementada sobre dispositivos reprogramables y que provea una eficiente capacidad de cálculo.

1.1.- OBJETIVOS

El objetivo general del proyecto es diseñar e implementar sobre dispositivos reprogramables, un sistema de compresión de imágenes que pueda ser adaptado a diversas aplicaciones, que permita tomar provecho de las tendencias tecnologías en dispositivos reprogramables y que provea una apreciable capacidad de computación.

Los objetivos específicos que se pretenden alcanzar son:

- Alcanzar un amplio conocimiento en las especificaciones de las técnicas de compresión usadas en la actualidad para definir su aplicabilidad dentro del sistema a desarrollar.
- Evaluar la adecuación de una tecnología flexible y de corto ciclo de diseño como son los dispositivos lógicos reprogramables a la solución del problema de la compresión de imágenes.
- Determinar una arquitectura adecuada, que permita un procesamiento eficiente al ser implementada sobre dispositivos reprogramables, y que sea adaptable para diversas aplicaciones.
- Implementar un prototipo funcional sobre dispositivos reprogramables.
- Evaluar la funcionalidad y capacidad del sistema implementado.
- Establecer un registro del desarrollo del proyecto que incluya información sobre las características del prototipo y los lineamientos que guíen a una mejora en trabajos futuros.

1.2.- ESTRUCTURA DEL LIBRO

Este documento se encuentra dividido en 6 capítulos los cuales se describen a continuación:

- En el capítulo dos se presentan las definiciones y conceptos fundamentales de la compresión de imágenes, así como una visión general de la estructura del sistema de compresión.

- En el capítulo tres se presenta una revisión conceptual de la transformada wavelet, partiendo de la teoría en el dominio continuo, para posteriormente pasar al dominio discreto, seguido de la transformación bidimensional y el análisis multiresolución a través de filtros digitales, también se realiza una revisión del estado del arte de las bases wavelet más utilizadas, seguido por un análisis que nos lleva a la selección de la que se considera más adecuada para la implementación sobre un dispositivo reprogramable, posteriormente se presenta la metodología seguida para el cálculo de los bits necesarios de acuerdo al nivel de transformación, finalmente se exponen algunos resultados sobre las pruebas del sistema.

- En el capítulo cuatro se presenta la arquitectura propuesta para la implementación de la Transformada Discreta Wavelet en dos dimensiones (2D-DWT) sobre una FPGAs. El capítulo inicia con una revisión del estado del arte sobre las arquitecturas hardware para la implementación de la 2D-DWT, seguido por un análisis desde una perspectiva de la implementación en una FPGA, posteriormente se presenta la arquitectura propuesta, finalmente se exponen algunos resultados de la puesta en práctica del módulo.

- En el capítulo cinco se presenta la etapa de compresión del sistema, el capítulo inicia con una revisión de algunas de las técnicas de compresión de imágenes más representativas, seguido por un análisis comparativo y finalmente se presenta una propuesta de modificación del algoritmo SPIHT (Set Partitioning in Hierarchical

Trees).

- En el capítulo seis se relacionan las pruebas realizadas sobre el sistema de compresión, el capitulo presenta pruebas de desempeño para diferentes niveles de transformación, para varios umbrales de cuantificación y finaliza con los resultados de las pruebas realizadas sobre una propuesta para codificación de video.

- El documento finaliza con el capítulo siete, en donde se presentan las conclusiones obtenidas a partir del desarrollo de este trabajo y los trabajos futuros propuestos para mejorar y dar continuidad a esta investigación.

Capítulo 2.

Sistema de Compresión de Imágenes

El objetivo general de un sistema de compresión es reducir el número de bits necesarios para representar la información con la menor distorsión posible, según lo mencionado en [2] la división más común de los métodos de compresión es: la compresión sin pérdida y la compresión con pérdida. En la compresión sin pérdidas, se condensan las cadenas de código sin despreciar nada de la información, por lo que la imagen se regenera intacta al ser descomprimida, sin embargo es menor la capacidad de compresión que provee este tipo de técnicas dado que su fin es permitir una recuperación exacta de la imagen. En la compresión con pérdidas los algoritmos usados para reducir las cadenas del código desechan información irrelevante y redundante de la imagen, los archivos comprimidos con este método pierden parte de los datos de la imagen basándose en el entendimiento del sistema de percepción visual humano, algunos algoritmos compensan esta pérdida con técnicas que hacen que la falta de información sea imperceptible a simple vista, este método permite un alto grado de compresión.

En [1] definen que un dato es irrelevante cuando su presencia no es perceptible por la mayoría de los observadores o cuando no produce efecto alguno sobre el sistema, por otro lado un dato es redundante cuando su presencia aunque perceptible no provee un aporte significativo a la información ya conocida. La redundancia puede ser espacial o temporal, la redundancia espacial esta formada por aquellos datos que son similares dentro de una cierta área de la imagen; la redundancia temporal esta formada por aquella información que se repite entre imágenes, campos o cuadros consecutivos. La redundancia espacial es explotada tanto por los sistemas de codificación de video como por los de imagen fija, la redundancia temporal es específica de la codificación de secuencias de video.

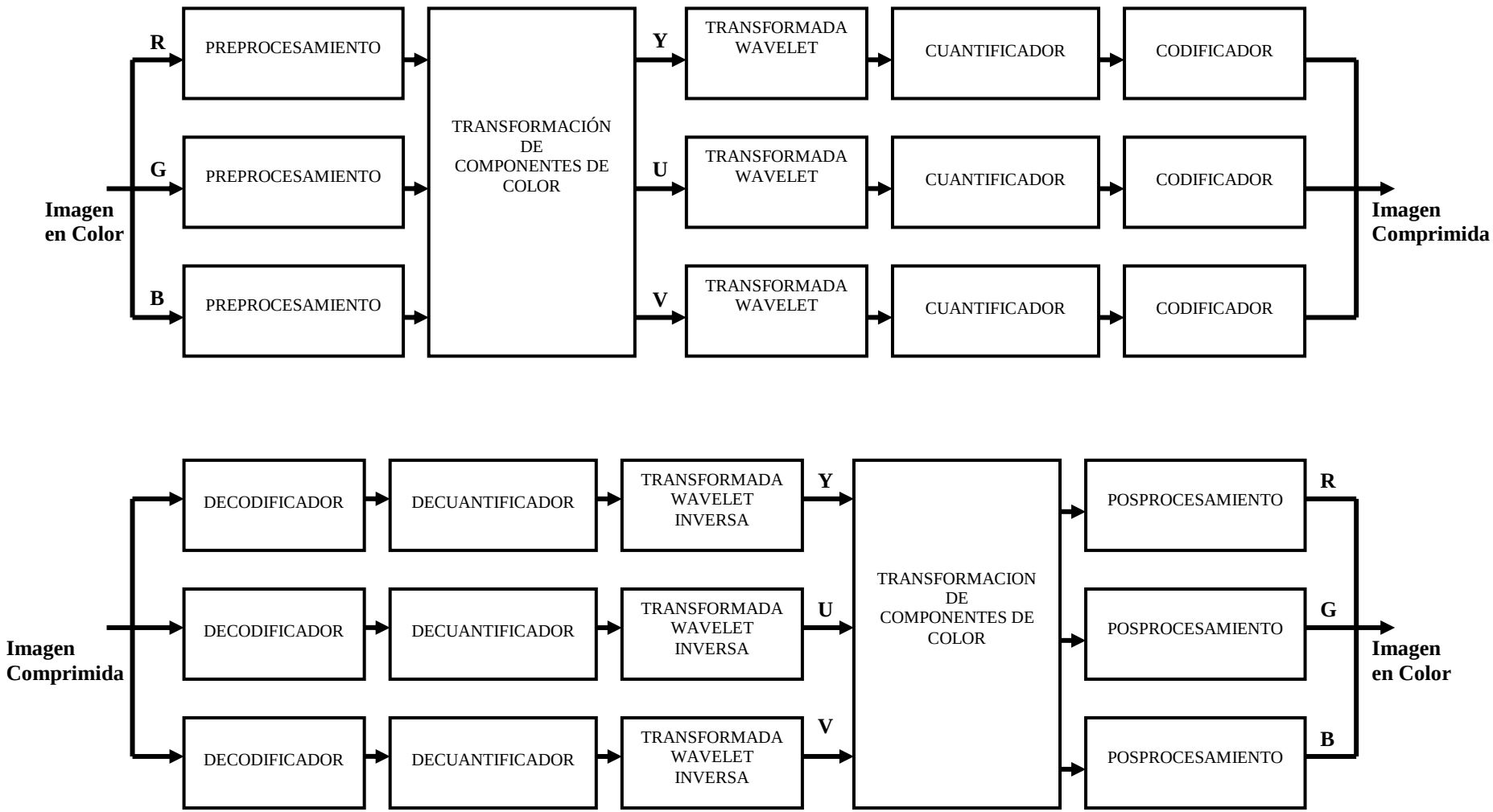


Figura 1.1 Esquema del Sistema de Compresión de Imágenes

2.1.- COMPONENTES DEL SISTEMA

La compresión de imágenes puede realizarse de diversos modos, el esquema desarrollado en este trabajo es mostrado en la figura 1.1, este sistema consta de cinco etapas y es similar al utilizado en JPEG2000 [3].

2.1.1.- Preprocesamiento - Posprocesamiento

Puesto que cada píxel de la imagen de entrada tiene valor entre 0 y 255 se realiza un reajuste del rango dinámico para que las muestras estén centradas alrededor de cero y tener una distribución de los valores que permita un mejor aprovechamiento del rango [47]. Este proceso se realiza solo si los datos de entrada son todos positivos, el ajuste del rango se realiza sustrayendo a cada muestra de entrada un valor de 2^{T-1} donde T es el número de bits que se usa para representar cada píxel, el proceso inverso debe realizarse en la reconstrucción, es decir, debe adicionarse un valor de 2^{T-1} a cada muestra de salida [3]. Este proceso se realiza en las etapas de preprocesamiento y posprocesamiento, ubicadas al inicio del compresor y al final del decompresor como se aprecia en la figura 1.1.

2.1.2.- Transformación de Componentes de Color

En el modelo de color RGB una imagen esta compuesta por tres arreglos rectangulares de muestras que especifican el color mediante cantidades positivas de los colores primarios rojo, verde y azul, un proceso de transformación se aplica sobre estos tres componentes para representar la imagen en el modelo de color YUV, dado que este modelo responde mejor a las características del sistema de visión humano [46] y permite reducir la correlación entre los componentes de la imagen para una compresión eficiente [3]. En el sistema propuesto cada componente de color es procesado de manera independiente, este proceso se realiza después del bloque de preprocesamiento y antes del posprocesamiento como se aprecia en

la figura 1.1. La transformación de color directa e inversa se realiza utilizando las siguientes ecuaciones [47]:

$$Y = \lfloor (R + 2G + B)/4 \rfloor \qquad G = Y - \lfloor (U + V)/4 \rfloor$$

$$U = B - G \qquad R = V + G \qquad \text{EC. 2.1}$$

$$V = R - G \qquad B = U + G$$

Al interior de la FPGA el preprocesamiento, posprocesamiento y la transformación de componentes de color se realizan a través de funciones desarrolladas en C++ que corren sobre el procesador Nios II [37], Nios II es un procesador definido en un lenguaje de descripción hardware que puede ser implementado sobre FPGAs de Altera, el código fuente puede ser consultado en el anexo C.

2.1.3.- Transformada Directa e Inversa

El uso de transformadas no reduce el numero de bits en la señal, sino que la transforman en una nueva señal que resulta más adecuada para su codificación con un menor número de bits por muestra; el éxito de la codificación de transformadas depende de qué tan bien la función base de la transformada represente las características de la señal de entrada, es importante que la trasformada concentre la energía de la imagen en unos pocos coeficientes para poder desechar el resto y que los coeficientes estén lo menos correlacionados unos de otros[4].

Una variedad de transformadas han sido usadas en codificación de imágenes, la transformada óptima que minimiza el error producido por el descarte de coeficientes es la transformada de Karhunen-Loeve [49], sin embargo su uso no es práctico ya que la matriz de transformación depende de la imagen por codificar y además su cálculo es laborioso, por eso se usa solo en planteamientos teóricos [5], en sistemas reales se emplean transformadas subóptimas entre las cuales las mas comunes son la transformada discreta de coseno (DCT) y la transformada discreta wavelet (DWT). Estas transformadas a pesar

de no ser óptimas poseen buenas propiedades de compresión de energía.

El esquema de compresión basado en DCT posee características como sencillez, buen desempeño y facilidad de implementación hardware para aplicaciones de propósito específico, pero debido a que la entrada necesita ser procesada por bloques la correlación a través de los límites de los bloques no es eliminada, se ha intentado reducir este problema usando un sobrelapamiento suave de los bloques pero esto aumenta la complejidad del algoritmo dando como resultado un incremento computacional que no justifica el ajuste del algoritmo [6]. Debido a que la DWT no necesita partir la imagen de entrada en bloques no presenta el problema de correlación en los límites de los bloques [51]; además estudios comparativos ente la DWT y la DCT revelan que la transformada wavelet presenta un mejor desempeño en sistemas de compresión [50] y su implementación hardware se puede realizar fácilmente de manera práctica a través de filtros [27], adicionalmente la transformación wavelet presenta otras ventajas como mayor robustez para la transmisión y decodificación de errores [6]. La descomposición de señales en distintas bandas de frecuencia realizada en el análisis wavelet es especialmente adecuada para la compresión, puesto que las imágenes naturales tienden a tener un espectro de frecuencia con la mayor parte de la energía concentrada en las frecuencias bajas de manera que se consigue una alta compactación de la energía, de otro lado de acuerdo con el sistema visual humano la visibilidad del ruido tiende a descender a altas frecuencias lo que permite a un diseñador ajustar el nivel de distorsión introducido por la compresión [51]. Por lo citado en este párrafo se decidió utilizar la transformada wavelet en el sistema de compresión desarrollado en este trabajo, los conceptos y técnicas usados para la transformación serán ampliados en el capítulo tres.

2.1.4.- Cuantificación - Decuantificación

La etapa de cuantificación restringe los valores de los coeficientes a un conjunto limitado y preferiblemente pequeño con lo que se reduce el número de bits necesarios para almacenar los coeficientes transformados y se disminuye la precisión de estos valores [6], el proceso

de cuantificación es la principal fuente de pérdidas en el compresor y aunque existen varias técnicas para la cuantificación la mayoría de los codificadores wavelet utilizan cuantificadores escalares por ser más simples en su implementación [4]. En el sistema propuesto los coeficientes transformados se cuantifican utilizando cuantificación escalar, matemáticamente el proceso de cuantificación y decuantificación es definido de acuerdo a las fórmulas [47]:

$$V(x,y) = sign(U(x,y)) \left\lfloor \frac{|U(x,y)|}{\Delta} \right\rfloor \qquad \text{E.C 2.2}$$

$$U(x,y) = (V(x,y) + r.signV(x,y))\Delta$$

Donde Δ es el tamaño del paso de cuantificación, U(x, y) es el valor del coeficiente transformado, V(x, y) es el coeficiente cuantificado y r es un sesgo. De acuerdo a lo recomendado en [3] para las pruebas del sistema el paso de cuantificación se estableció en uno, pero se dejo como un parámetro modificable para aplicaciones posteriores.

Al interior de la FPGA la cuantificación y decuantificación también se realizan a través de funciones desarrolladas en C++ que corren sobre el procesador Nios II, el código fuente puede ser consultado en el anexo C.

2.1.5.- Codificador - Decodificador

El diseño de codificadores es una parte fundamental en el sistema ya que tiene la tarea de comprimir los coeficientes cuantificados, la selección del codificador usualmente depende de los requerimientos de memoria, la velocidad de ejecución, el ancho de banda disponible y la calidad en la reconstrucción de la imagen. Existe una gran variedad de esquemas de codificación, entre los cuales los más mencionados son los codificadores de entropía y los codificadores en árbol; entre los esquemas de codificación de imágenes basados en wavelet se destacan EZW (Embedded Zerotree Wavelet), SPIHT (Set Partitioning in Hierarchical Trees), EBCOT (Embedded Block Coding whit Optimal Truncation Points) y

LTW (Wavelet Lower Trees). En el capitulo cinco se amplían las técnicas de codificación y se detalla el algoritmo utilizado en la implementación del sistema de compresión.

.

Capítulo 3.

Transformada Wavelet

Como se menciona anteriormente la transformada wavelet es una de las más usadas en la compresión de imágenes, por tal motivo en este capítulo se presenta una revisión conceptual de la transformada wavelet, partiendo de la teoría en el dominio continuo, para posteriormente pasar al dominio discreto, seguido de la transformación bidimensional y el análisis multiresolución a través de filtros digitales. También se realiza una revisión del estado del arte de las bases wavelet más utilizadas, seguido por un análisis que nos lleva a la selección de la que se considera más adecuada para la implementación sobre un dispositivo reprogramable, posteriormente se presenta la metodología seguida para el cálculo de los bits necesarios de acuerdo al nivel de transformación, finalmente se exponen algunos resultados sobre las pruebas del sistema.

3.1.- TRANSFORMADA WAVELET CONTINUA.

El propósito de esta sección es describir las características fundamentales de la teoría asociada a la transformada wavelet [7] [8] [9] [10] [11].

Las wavelet son familias de funciones que se emplean para el análisis de otras funciones, dado que permiten representar una señal como una descomposición de señales más sencillas, a través de ellas es posible obtener una descripción tiempo-escala de una señal. La representación de señales por descomposición no es nueva, aproximadamente hace 200 años Joseph Fourier planteó la representación de funciones mediante la superposición de

senos y cosenos, la idea ha evolucionado con el tiempo y las investigaciones más recientes nos llevan a otro tipo de transformaciones entre las cuales encontramos las Wavelets.

El análisis wavelet se realiza partiendo de la definición de una función principal llamada wavelet madre, que se dilata y traslada para definir una base o familia de funciones de wavelet, de la siguiente manera [11]:

$$\psi_{a,b}(t) = |a|^{-1/2}\psi\left(\frac{t-b}{a}\right); a,b \in R, a \neq 0 \qquad \text{EC 3.1}$$

Donde la wavelet madre esta representada por $\psi(t)$, el parámetro a representa la escala y b representa el desplazamiento o traslación. Al variar el parámetro a se dilata o se contrae la señal, como consecuencia se reduce o aumenta la frecuencia de la señal en análisis, de esta manera se obtiene información referente a los componentes de frecuencia; el parámetro de traslación nos permite ubicar un lugar especifico en el tiempo.

La transformada continua wavelet DWT unidimensional de una señal está definida [9] por la ecuación:

$$W(a,b) = |a|^{-1/2}\int_{-\infty}^{\infty} f(t)\psi^*\left(\frac{t-b}{a}\right)dt \qquad \text{EC 3.2}$$

Donde el ψ^* representa el complejo conjugado en el caso de que $\psi(t)$ tenga valores complejos, $\psi_{a,b}$ es la familia wavelet y $f(t)$ es la función a analizar.

En la transformación wavelet para que la reconstrucción sea posible se debe garantizar la siguiente condición [9]:

$$C_\psi = \int_{-\infty}^{\infty}\frac{|\Psi(\omega)|^2}{|\omega|}d\omega < \infty \qquad \text{EC 3.3}$$

Donde $\Psi(\omega)$ es la transformada de fourier de la wavelet. Esta condición es conocida como la *condición de admisibilidad*.

La función f puede ser recuperada desde su transformada a través de la fórmula de reconstrucción, definida en la siguiente ecuación [11]:

$$f(t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}W(a,b)\psi_{a,b}\frac{dadb}{a^2} \qquad \text{EC 3.4}$$

3.2.- TRANSFORMADA WAVELET DISCRETA.

Para el análisis de señales digitales, se hace necesario implementar una versión discreta. La selección de los valores discretos para a y b debe realizarse de manera conveniente para obtener características apropiadas en el sistema de muestreo, como muestreo fino para altas frecuencias, muestreo grueso para bajas frecuencias, acople al sistema de percepción humano y cálculo eficiente. En [11] se plantea:

$$a = a_0^{\,m} \quad \text{y} \quad b = nb_0a_0^{\,m} \quad \text{donde} \quad m,n \in Z \ \text{y} \ a_0 > 1 \quad b_0 > 0 \qquad \text{EC 3.5}$$

La apropiada selección de a_0 y b_0 depende de la wavelet utilizada. De acuerdo a lo anterior se expresa la base wavelet como sigue:

$$\psi_{m,n}(x) = a_0^{-m/2}\psi(a_0^{-m}x - nb_0) \qquad \text{EC.3.6}$$

En concordancia, la transformada discreta wavelet (DWT) de una señal $f(x)$ se define como [11]:

$$W(a_m, b_n) = a_0^{-m/2}\int_{-\infty}^{\infty} f(x)\psi(a_0^{-m}x - nb_0)dx \qquad \text{EC 3.7}$$

La multiresolución es una técnica muy utilizada en el análisis de señales, consiste en obtener una representación simultanea de la señal a diferentes resoluciones, mediante la dilatación y traslación de una función de escalamiento y la función wavelet asociada; uno de los grandes avances en el análisis wavelet fue su implementación usando filtros, los cuales fueron formados utilizando los coeficientes de las funciones de escalamiento y wavelet [12]. El filtro de paso bajo esta asociado a la función de escalamiento, la señal obtenida a su salida es una versión suavizada de baja resolución de la señal original; el filtro de paso alto esta asociado a la función wavelet, la señal obtenida a su salida contiene los detalles de la señal; la transformación esta definida por las siguientes ecuaciones [13]:

$$A[x] = \sum_k h[k]f[2x-k] \qquad \text{EC 3.8}$$

$$D[x] = \sum_k g[k]f[2x-k] \qquad \text{EC 3.9}$$

Donde f es la señal original, g y h son los coeficientes de los filtros de paso alto y de paso bajo, A y D son los coeficientes wavelets de aproximación y detalle. La idea general es convolucionar la secuencia de entrada con los filtros y submuestrear por dos para eliminar la redundancia; este proceso se ilustra en el diagrama de bloques de la figura 3.1.

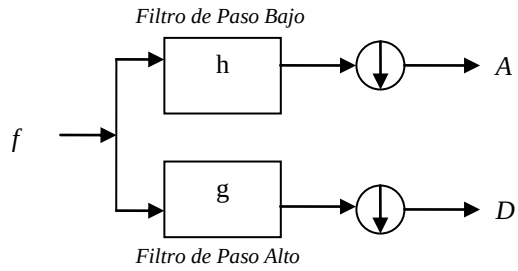


Figura 3. 1 Diagrama de bloques del banco de filtros de la DWT para un nivel

Para lograr reconstruir la señal original a partir de sus coeficientes wavelets, es necesario que los filtros cumplan algunas consideraciones [9] [14] de manera que sean complementarios y la suma de *A(x)* y *D(x)* sea *f (x)*; si se diseñaran los filtros en forma muy separada se perdería información, o en caso contrario se estaría amplificando la banda de entrecruzamiento.

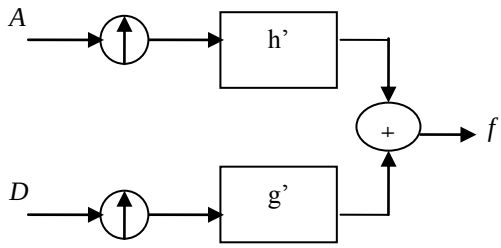


Figura 3. 2 Esquema de reconstrucción Wavelet para un nivel

La figura 3.2 ilustra el proceso de cálculo de la Transformada Inversa de Wavelet Discreta (IDWT), donde *h'* y *g'* son complementarios de *h* y *g*, en este caso debe realizarse una sobrerepresentación de las muestras para compensar el submuestreo realizado en el proceso de descomposición, luego pasa por un proceso de filtrado, para finalmente reconstruir f .

Para análisis mas complejos de señales, no basta con dos bandas de frecuencias (alta y baja), en estos casos debe hacerse una descomposición de más niveles para poder separar las características y analizarlas de manera independiente, la elección del número de niveles

depende de la naturaleza de la señal y de la aplicación específica. Para la realización de filtros multinivel se puede iterar el proceso de filtrado, es decir, aplicar el mismo procedimiento a las señales de salida de la primera etapa, y así sucesivamente hasta el nivel de precisión que se desee, este esquema es conocido como algoritmo de Mallat. La idea se ilustra en la figura 3.3 donde se muestra el diagrama de bloques del banco de filtros de la DWT para tres niveles [15].

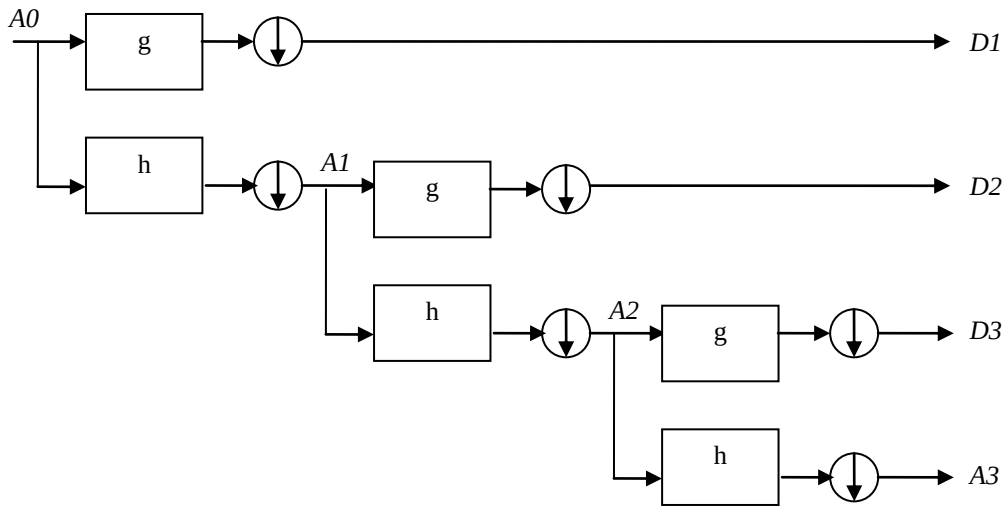


Figura 3. 3 Diagrama de bloques del banco de filtros de la DWT para tres niveles

Teniendo en cuenta el nivel de iteración, la transformada es definida por las siguientes ecuaciones [13]:

$$A[x]_{J+1} = \sum_k h[k] A_J[2x-k] \qquad \text{EC 3.10}$$

$$D[x]_{J+1} = \sum_k g[k] A_J[2x-k] \qquad \text{EC 3.11}$$

Donde J es el nivel de aplicación de la transformada y $A[x]_0$ es la señal original.

3.3.- TRANSFORMADA WAVELET DISCRETA EN DOS DIMENSIONES (2D-DWT).

La transformada unidimensional definida anteriormente se puede extender a funciones bidimensionales, en general el análisis wavelet puede ser ampliado a cualquier dimensión,

para el caso de las imágenes trabajaremos en dos dimensiones.

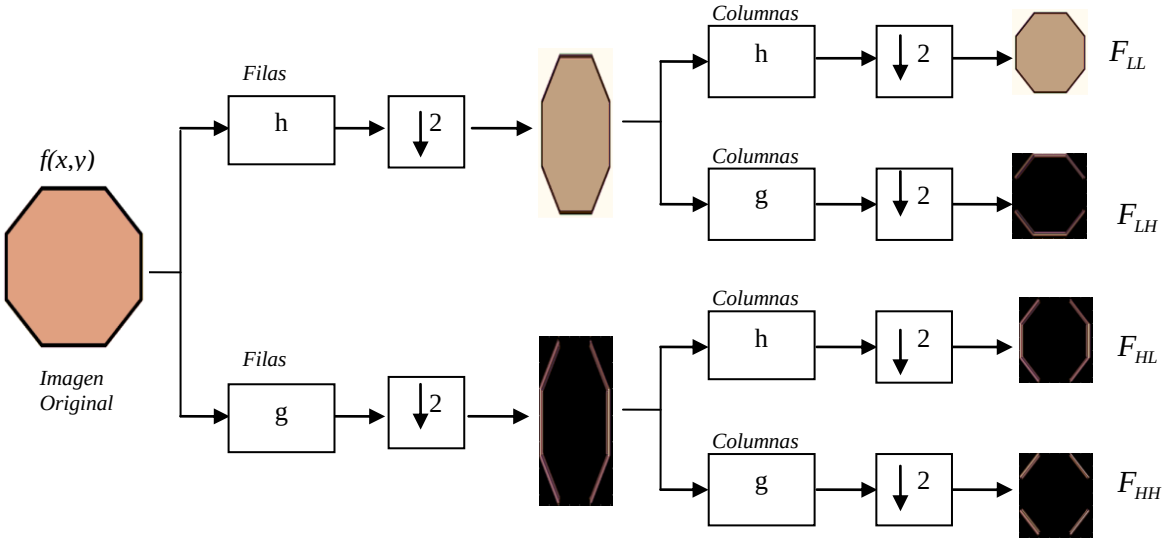
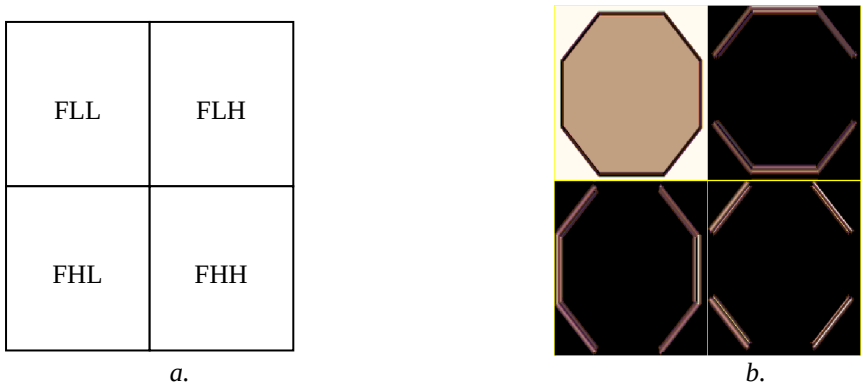


Figura 3. 4 Diagrama de bloques del banco de filtros utilizado para calcular la 2D-DWT

La figura 3.4 ilustra el desarrollo de la transformada wavelet de una imagen utilizando filtros unidimensionales. Primero se desarrolla la transformación wavelet sobre cada una de las filas de la imagen, lo que genera dos imágenes intermedias que representan la aproximación y el detalle sobre el eje x, posteriormente se realiza la transformación sobre cada columna de las imágenes intermedias, como resultado se obtiene una versión suavizada de la imagen o promedio F_{LL} y tres subimágenes con los detalles F_{LH}, F_{HL}, F_{HH}. F_{LH} enfatiza las característica horizontales, F_{HL} las verticales y F_{HH} las diagonales [16].

Para el análisis de imágenes, la transformada wavelet es a menudo representada como se ilustra en la figura 3.5 [17].



a. *b.*

Figura 3. 5 (a) Descomposición wavelets de una imagen a un nivel (b) Descomposición de la imagen Hexágono.

La transformación puede ser definida de acuerdo con las expresiones [13]:

$$F_{LL}(x,y) = \sum_{k1}\sum_{k2} h(k_1)h(k_2)f(2x-k_1,2y-k_2) \qquad \text{EC 3.12}$$

$$F_{LH}(x,y) = \sum_{k1}\sum_{k2} h(k_1)g(k_2)f(2x-k_1,2y-k_2) \qquad \text{EC 3.13}$$

$$F_{HL}(x,y) = \sum_{k1}\sum_{k2} g(k_1)h(k_2)f(2x-k_1,2y-k_2) \qquad \text{EC 3.14}$$

$$F_{HH}(x,y) = \sum_{k1}\sum_{k2} g(k_1)g(k_2)f(2x-k_1,2y-k_2) \qquad \text{EC 3.15}$$

La figura 3.6 ilustra el proceso de cálculo de la Transformada Inversa de Wavelet Discreta en dos dimensiones (2D-IDWT), donde *h'* y *g'* son complementarios de *h* y *g*, al igual que en el caso unidimensional debe realizarse una sobrerepresentación de las muestras para compensar el submuestreo realizado en el proceso de descomposición, luego pasa por un proceso de filtrado, para finalmente reconstruir la imagen.

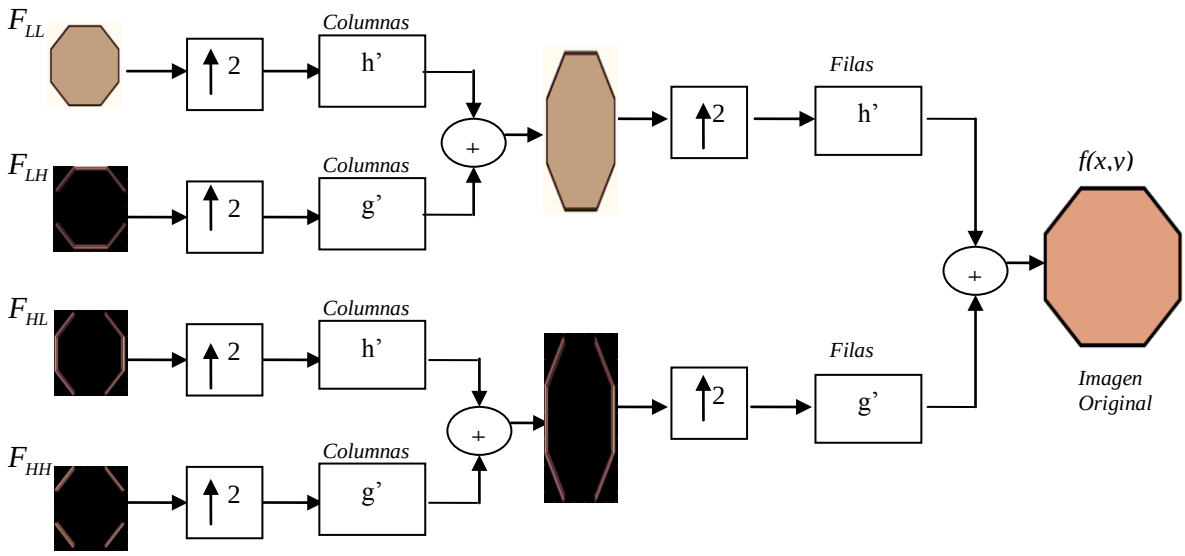


Figura 3. 6 Diagrama de bloques del banco de filtros utilizado para calcular la 2D-DWT

Al igual que en la transformación en una dimensión el proceso se puede iterar para niveles superiores, asumiendo la imagen promedio F_{LL} como entrada del siguiente nivel; la figura 3.7 ilustra la transformación para dos niveles.

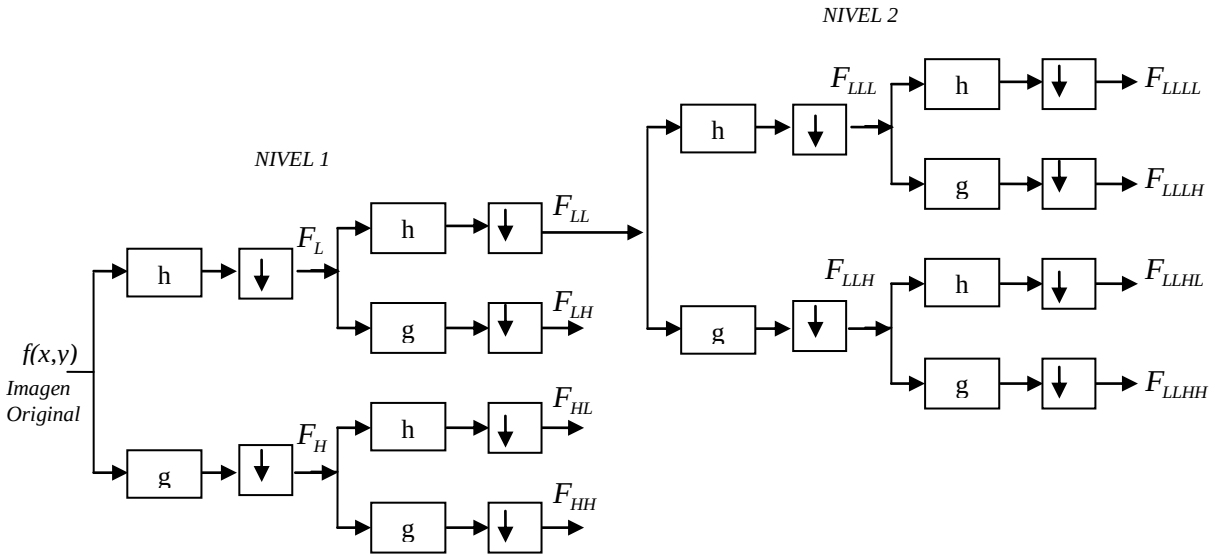


Figura 3. 7 Diagrama de bloques del banco de filtros utilizado para calcular dos niveles de la 2D-DWT

3.4.- BASE WAVELET, ESTADO DEL ARTE

La selección de la base wavelet es uno de los parámetros más importantes en la compresión de imágenes puesto que el resultado cambia dependiendo de ésta selección. Existen numerosas publicaciones [14] [18] [19] [20] [21] [22] [23] [12] que presentan criterios para seleccionar la wavelet base como suavidad, momentos de desvanecimiento, regularidad, medidas de comportamiento oscilatorio, simetría, soporte compacto, sensibilidad, precisión de la aproximación, número de coeficientes, selectividad en frecuencia, etc.; sin embargo, la mejor combinación de estas características no es un criterio unificado, teniendo en cuenta que uno de los objetivos de este trabajo es realizar la implementación sobre un dispositivo reprogramable es necesario considerar un esquema que permita tomar provecho de esta tecnología.

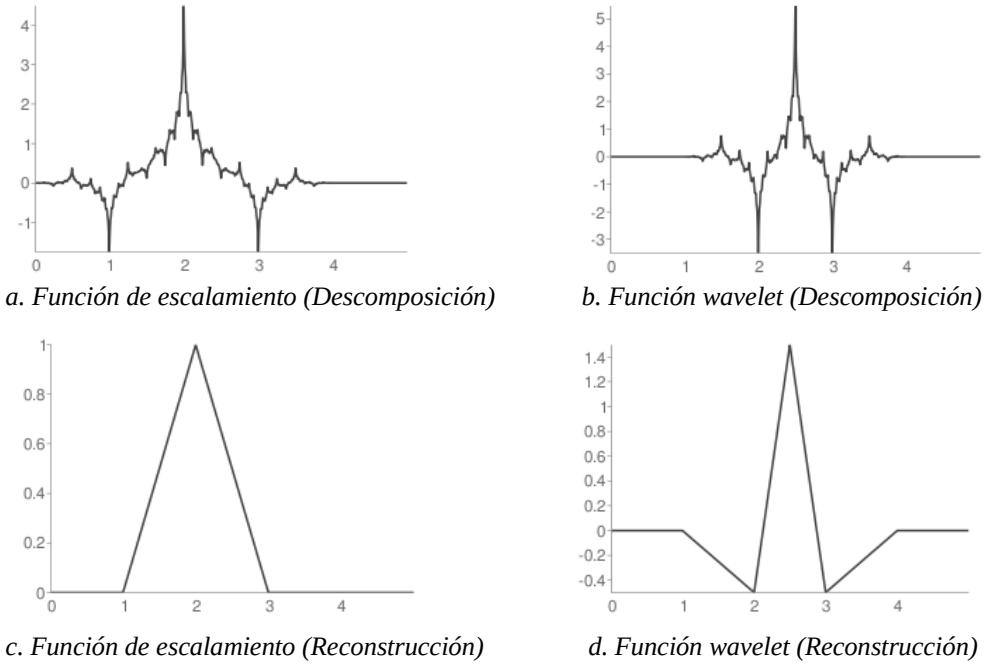
En la investigación realizada en [19] señalan la importancia de la regularidad y los momentos de desvanecimiento para la codificación de imágenes, como resultado del análisis realizado los autores consideran el filtro biortogonal spline variante (9,7) como uno de los más adecuados en aplicaciones de codificación de imágenes, en [22] ratifican esta observación presentándolo como el mejor de los filtros estudiados y uno de los mas usados.

En [12] comparan el desempeño de la compresión de las wavelets de daubechies, haar, coiflet y biortogonales usando medidas de comparación como la relación señal a ruido y la calidad de la imagen, finalmente concluyen que la wavelet biortogonal (5,3) (también conocida como CDF(2.2)) es la que provee la mejor calidad visual. En [23] realizan una comparación entre las bases wavelets mas utilizadas para determinar la mas apropiada para la transformación de imágenes a color, los autores presentan las wavelet de daubechies 4 y biortogonal (9,3) (también conocida como BW2.4) como las bases con mejores resultados para las imágenes naturales. En [18] realizan una amplia evaluación de los filtros biortogonales con un número de coeficientes inferior a 36, en el análisis se consideran criterios de evaluación como la regularidad, la relación señal a ruido, la respuesta al impulso y la respuesta al escalón, como resultado de esta investigación los autores presentan la lista con los seis filtros biortogonales mas destacados (bior (9,7), bior (13,11), bior (6,10), bior (5,3), bior (2,6) y bior (9,3)).

3.5.- SELECCIÓN DE LA BASE WAVELET

Todas las bases wavelet mencionadas en la sección anterior poseen características que las hacen apropiadas para la utilización en sistemas de codificación; sin embargo, para seleccionar la base wavelet más conveniente en la implementación del codificador sobre un dispositivo reprogramable es necesario considerar el número de elementos lógicos necesarios para la implementación, debido a que el área del dispositivo es un recurso limitado. Se consideraron las bases Bior5.3 y Bior9.3 debido a que además de poseer características que las ubican entre las más recomendadas para aplicaciones de codificación de imágenes, los coeficientes de estas bases pueden ser transformados a números enteros mediante una simple normalización por un factor de $\sqrt{2}$, por lo tanto no se requiere utilizar en la implementación módulos con aritmética en punto flotante, de esta manera se disminuye el área necesaria para la implementación; un segundo criterio de selección es el número de coeficientes de la base, con menos coeficientes se requieren menos operaciones para implementar la transformación y por ende se utiliza menos área; teniendo en cuenta

que la base Bior5.3 posee menos coeficientes que la Bior9.3, en la presente aplicación se utilizará la base Bior5.3, la figura 3.8 ilustra las funciones asociadas a esta base.



a. Función de escalamiento (Descomposición)

b. Función wavelet (Descomposición)

c. Función de escalamiento (Reconstrucción)

d. Función wavelet (Reconstrucción)

Figura 3. 8 Base Biortogonal (5,3) [1]

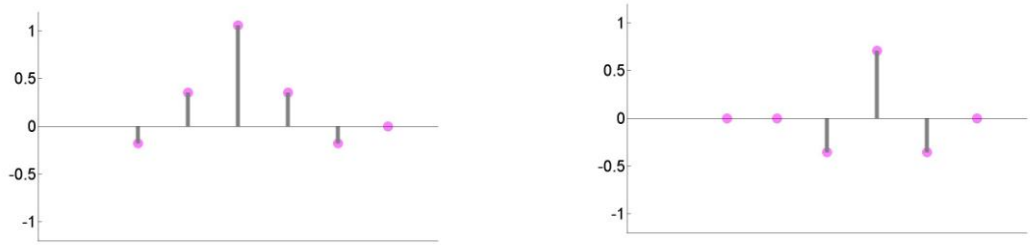
Los coeficientes usados para la implementación de la etapa de descomposición usando la base biortogonal (5,3), a través de filtros, se listan en la tabla 3.1 y se grafican en la figura 3.9.

Tabla 3. 1 Coeficientes para la descomposición usando la Base Biortogonal (5,3)

n	**-2**	**-1**	**0**	**1**	**2**	**3**
Filtro de Paso Bajo h(n)	$-1\frac{\sqrt{2}}{8}$	$2\frac{\sqrt{2}}{8}$	$6\frac{\sqrt{2}}{8}$	$2\frac{\sqrt{2}}{8}$	$-1\frac{\sqrt{2}}{8}$	
	-0.17677	0.35355	1.06066	0.35355	-0.17677	

[1] Imagen tomada de http://wavelets.pybytes.com/wavelet/bior2.2/

Filtro de Paso Alto g(n)			$-2\frac{\sqrt{2}}{8}$	$4\frac{\sqrt{2}}{8}$	$-2\frac{\sqrt{2}}{8}$	
			-0.35355	0.70710	-0.35355	



a. Función de escalamiento (Filtro de Paso Bajo) *b. Función wavelet (Filtro de Paso Alto)*

Figura 3. 9 Coeficientes para la descomposición usando la Base Biortogonal (5,3)

De la teoría de bancos de filtros para tener una reconstrucción perfecta de la imagen, los coeficientes de los filtros deben cumplir [18]:

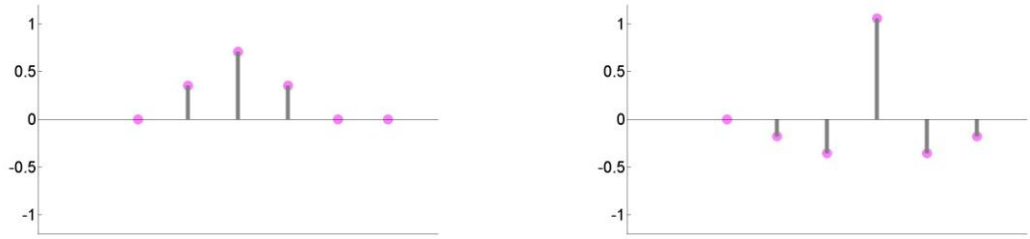
$$g'(n) = (-1)^{n+1} h(n) \qquad \text{EC 3.16}$$

$$g(n) = (-1)^{n} h'(n) \qquad \text{EC 3.17}$$

Por lo tanto, los coeficientes usados para la implementación de la etapa de reconstrucción usando la base biortogonal (5,3), a través de filtros, se listan en la tabla 3.2 y se grafican en la figura 3.10.

Tabla 3. 2 Coeficientes para la reconstrucción usando la Base Biortogonal (5,3)

n	**-2**	**-1**	**0**	**1**	**2**	**3**
Filtro de Paso Bajo h'(n)		$2\frac{\sqrt{2}}{8}$	$4\frac{\sqrt{2}}{8}$	$2\frac{\sqrt{2}}{8}$		
		0.35355	0.70710	0.35355		
Filtro de Paso Alto g'(n)		$-1\frac{\sqrt{2}}{8}$	$-2\frac{\sqrt{2}}{8}$	$6\frac{\sqrt{2}}{8}$	$-2\frac{\sqrt{2}}{8}$	$-1\frac{\sqrt{2}}{8}$
		-0.17677	-0.35355	1.06066	-0.35355	-0.17677



a. Función de escalamiento (Filtro de Paso Bajo) *b. Función wavelet (Filtro de Paso Alto)*

Figura 3. 10 Coeficientes para la reconstrucción usando la Base Biortogonal (5,3)

3.6.- ANCHO DE BITS DE LOS COEFICIENTES

El filtrado recursivo de los datos de la imagen de entrada produce un crecimiento en el ancho de bits necesarios para representar los coeficientes de la transformación. En diseños hardware entre mayor sea el número de bits, se requiere de un área mayor para la implementación. Por esta razón es importante conocer el mínimo tamaño necesario para los datos sin que esto implique pérdida de información.

Definimos *TB* como el tamaño en bits de los píxeles de la imagen de entrada; si usamos una representación en complemento a dos, los posibles valores del píxel están en el intervalo $\left[-2^{TB-1}, 2^{TB-1}-1\right]$; para el cálculo se utiliza la base Bior5.3 normalizada en un factor de $\frac{\sqrt{2}}{8}$, los valores mínimos y máximos de los coeficientes a la salida del filtrado por filas del primer nivel serán:

Valor Máximo, obtenido a la salida del filtro de paso bajo del primer nivel procesado por filas:

$$V_{\max F}h = -1\left(-2^{TB-1}\right) + 2\left(2^{TB-1}-1\right) + 6\left(2^{TB-1}-1\right) + 2\left(2^{TB-1}-1\right) - 1\left(-2^{TB-1}\right) = 12\left(2^{TB-1}\right) - 10$$

Valor Mínimo, obtenido a la salida del filtro de paso bajo del primer nivel procesado por filas:

$$V_{\min F}h = -1(2^{TB-1}-1) + 2(2^{TB-1}) + 6(2^{TB-1}) + 2(2^{TB-1}) - 1(2^{TB-1}-1) = -12(2^{TB-1}) + 2$$

Valor Máximo, obtenido a la salida del filtro de paso alto del primer nivel procesado por filas:

$$V_{\max F}g = -2(2^{TB-1}) + 4(2^{TB-1}-1) - 2(2^{TB-1}) = 8(2^{TB-1}) - 4$$

Valor Mínimo, obtenido a la salida del filtro de paso alto del primer nivel procesado por filas:

$$V_{\min F}g = -2(2^{TB-1}-1) + 4(2^{TB-1}) - 2(2^{TB-1}-1) = -8(2^{TB-1}) + 4$$

Por consiguiente los valores de los coeficientes del filtrado por filas del primer nivel están en el rango $[-12x2^{TB-1}+2, 12x2^{TB-1}-10]$ para los coeficientes generados a la salida del filtro de paso bajo y $[-8x2^{TB-1}+4, 8x2^{TB-1}-4]$ para los coeficientes generados a la salida del filtro de paso alto. Estos son los rangos de entrada a la etapa de filtrado por columnas que constituye la salida del primer nivel, en esta etapa también se usa la base Bior5.3, pero se denormaliza multiplicando por un factor de $1/32$; los valores mínimos y máximos del primer nivel serán:

Valor Máximo, obtenido a la salida del filtro de paso bajo del primer nivel:

$$V_{\max}h = -1(-12x2^{TB-1}+2)/32 + 2(2x2^{TB-1}-10)/32 + 6(2x2^{TB-1}-10)/32 + 2(2x2^{TB-1}-10)/32 - 1(-12x2^{TB-1}+2)/32$$

Valor Mínimo, obtenido a la salida del filtro de paso bajo del primer nivel:

$$V_{\min}h = -1(2x2^{TB-1}-10)/32 + 2(-12x2^{TB-1}+2)/32 + 6(-12x2^{TB-1}+2)/32 + 2(-12x2^{TB-1}+2)/32 - 1(2x2^{TB-1}-10)/32$$

Valor Máximo, obtenido a la salida del filtro de paso alto del primer nivel:

$$V_{\max}g = -2(-12x2^{TB-1}+2)/32 + 4(2x2^{TB-1}-10)/32 - 2(-12x2^{TB-1}+2)/32$$

Valor Mínimo, obtenido a la salida del filtro de paso alto del primer nivel:

$$V_{\min}g = -2(2x2^{TB-1}-10)/32 + 4(-12x2^{TB-1}+2)/32 - 2(2x2^{TB-1}-10)/32$$

Para obtener el rango de los siguientes niveles, se utiliza el rango calculado para el nivel anterior y los coeficientes de la base Bior5.3, de manera análoga como se realizó para el primer nivel.

La tabla 3.3 muestra el resultado del cálculo realizado para obtener el rango de los coeficientes para varios niveles, se asume que el tamaño de los píxeles de entrada es de 9 bits. Se realizó un programa en Matlab para obtener los datos, el código utilizado se presenta en el anexo A.

Tabla 3. 3 Ancho de bit de acuerdo al nivel de transformación

Nivel	$V_{\min}h$	$V_{\max}h$	$V_{\min}g$	$V_{\max}g$	*Ancho bit h*	*Ancho bit g*
1	-575	573	-383	383	11	10
2	-2585	2581	-1722	1722	13	12
3	-11628	11620	-7749	7749	15	14
4	-52316	52300	-34872	34872	17	17
5	-235402	235370	-156924	156924	19	19
6	-1059269	1059205	-706158	706158	22	21

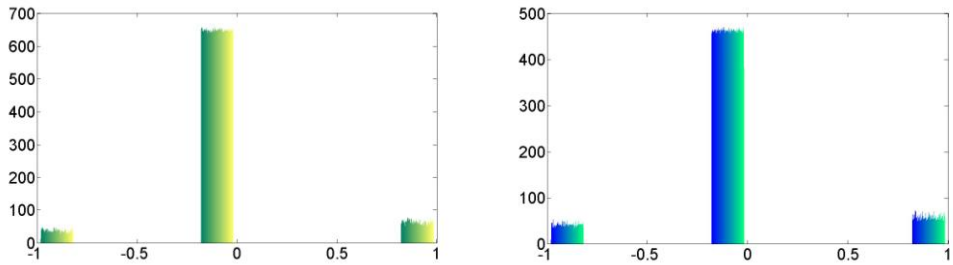
3.7.- VERIFICACIÓN DEL SISTEMA

Se realizaron pruebas en Matlab para verificar y probar que al normalizar y denormalizar la base biortogonal (5,3) mediante un factor de $\frac{\sqrt{2}}{8}$; es posible usar coeficientes enteros con 16 bits de precisión durante la transformación, sin afectar considerablemente la calidad en la reconstrucción de la imagen. El código utilizado para la realización de estas pruebas se presenta en el anexo A.

En la Tabla 3.4 se presentan los resultados de transformar y reconstruir algunas imágenes de prueba, usando para la transformación coeficientes enteros y coeficientes en doble punto flotante; una mirada rápida de estas imágenes indica que no existen grandes diferencias, sin embargo es necesario presentar medidas cuantitativas que determinen el grado de variación y si este es fundamental en el desarrollo de la aplicación. Un indicador usado para determinar la magnitud del error es la RMSE (raíz del error cuadrático medio), dicho error se calcula sumando el cuadrado del error, es decir, de la diferencia entre cada píxel de la imagen original *f* y la final *f* ‘, donde *N* es el número total de píxeles [24]:

$$RMSE = \sqrt{\frac{1}{N}\sum_{x,y}\left[f(x,y) - f'(x,y)\right]^2} \qquad \text{EC 3. 18}$$

El RMSE entre las reconstrucciónes obtenidas de transformaciones en números enteros y en punto flotante de 5 imágenes de prueba se presenta en la ultima columna de la tabla 3.4, podemos observar que de acuerdo a lo planteado en [25] el valor del error es aceptable, puesto que RMSE<1/2. Los resultados mostrados en la tabla 3.4 pueden ser complementados al observar la figura 3.11, en la cual se grafica el histograma de la matriz de diferencia entre las reconstrucción obtenida a través del uso de filtros con coeficientes enteros y la imagen original, para las imágenes de prueba 1 y 5; se nota que la mayor parte de los píxeles se reconstruyen perfectamente y que en los píxeles en los cuales se presenta diferencia, esta no es mayor a 1 en la escala de 255 niveles de gris. Por lo anterior podemos inducir que la calidad de la reconstrucción de la imagen no se afectará considerablemente al usar coeficientes enteros.



a. Matriz Diferencia-Imagen Prueba 1 *b. Matriz Diferencia-Imagen Prueba 5*

Figura 3. 11 Histograma de la matriz de diferencia entre las reconstrucciones obtenidas a través del uso de filtros con coeficientes enteros y la imagen original

Tabla 3. 4 RMSE entre las imágenes reconstruidas a través de coeficientes enteros y coeficientes con doble punto flotante

<table>
<tr><th>IMAGEN ORIGINAL</th><th>IMAGEN RECONSTRUIDA</th><th>RMSE</th></tr>
<tr><td rowspan="2"> 
Imagen de Prueba 1</td><td> 
Coeficientes enteros</td><td rowspan="2">0.3021</td></tr>
<tr><td> 
Coeficientes dobles</td></tr>
<tr><td rowspan="2"> 
Imagen de Prueba 2</td><td> 
Coeficientes enteros</td><td rowspan="2">0.3118</td></tr>
<tr><td>  </td></tr>
</table>

<table>
<tr><td></td><td>Coeficientes dobles</td><td></td></tr>
<tr><td rowspan="2"> 
Imagen de Prueba 3</td><td> 
Coeficientes enteros</td><td rowspan="2">0.3119</td></tr>
<tr><td> 
Coeficientes dobles</td></tr>
<tr><td rowspan="2"> 
Imagen de Prueba 4</td><td> 
Coeficientes enteros</td><td rowspan="2">0.3368</td></tr>
<tr><td>  </td></tr>
</table>

<table>
<tr><td></td><td>Coeficientes dobles</td><td></td></tr>
<tr><td rowspan="2"> 
Imagen de Prueba 5</td><td> 
Coeficientes enteros</td><td rowspan="2">0.3454</td></tr>
<tr><td> 
Coeficientes dobles</td></tr>
</table>

Capítulo 4.

Arquitecturas Hardware Para la Implementación de la Transformada Discreta Wavelet en Dos Dimensiones

En este capítulo se presenta la arquitectura propuesta para la implementación de la Transformada Discreta Wavelet en dos dimensiones (2D-DWT) sobre una FPGA; en el diseño se buscó un balance entre número de celdas lógicas requeridas y la velocidad de procesamiento.

El capítulo inicia con una revisión del estado del arte sobre las arquitecturas hardware para la implementación de la 2D-DWT seguido por un análisis desde una perspectiva de la implementación en una FPGA, posteriormente se presenta la arquitectura propuesta, finalmente se exponen algunos resultados de la puesta en práctica del módulo.

En concordancia con lo presentado en el capítulo 2, para las arquitecturas presentadas se utiliza J para indicar el nivel de transformación, K para el tamaño del filtro y la imagen a procesar es de tamaño de NxN.

4.1.-IMPLEMENTACIONES HARDWARE, ESTADO DEL ARTE

Existen numerosos estudios [26] [27] [13] [30] [31] [32] que desarrollan arquitecturas para la implementación de la 2D-DWT, en esta sección se presentará una breve descripción de algunas de las arquitecturas utilizadas resaltando las principales características.

Chen *et al* [26] proponen una arquitectura que calcula la 2D-DWT mediante una adaptación del Algoritmo Piramidal Recursivo (RPA). El RPA propuesto por Vishwanath [28] es una reformulación del algoritmo piramidal propuesto por Mallat [29]. La idea general para una dimensión, es replantear el orden en el cual se calculan los coeficientes de la transformada de forma tal que pueda iniciarse el cálculo del siguiente nivel sin completar el cálculo de los coeficientes del nivel anterior. Chen presenta una adaptación para transformaciones en dos dimensiones y en lugar de organizar la transformación realizando el cálculo píxel a píxel se planifica por filas, por consiguiente se deben calcular en paralelo los coeficientes de toda una fila. La Figura 4.1 muestra la arquitectura para procesamiento paralelo por filas. La arquitectura tiene cinco componentes fundamentales: los procesadores elementales (PEs), la red de conexión, las celdas de memoria, los elementos de direccionamiento y el controlador. La entrada de datos se realiza de manera serial en un buffer, cuando la fila está completa se almacena en memoria; la red de conexión lleva los datos de toda una fila hacia los PEs, donde se realiza el cálculo de los coeficientes; es el controlador quien planifica, siguiendo el RPA, que datos deben llegar a los PEs. Esta arquitectura requiere de memoria de tamaño $(K+1)NJ$ y utiliza N^2+N ciclos de reloj para procesar una imagen de tamaño NxN. El esfuerzo requerido para mantener el rastro de los últimos coeficientes calculados, aumenta la complejidad del controlador convirtiéndose en la principal desventaja del RPA.

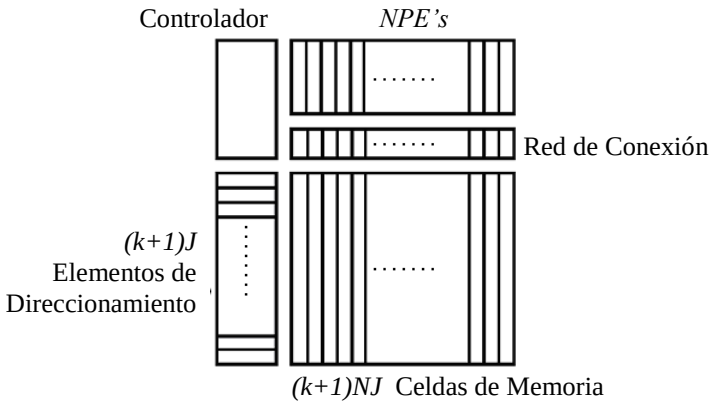


Figura 4. 1 Arquitectura de Procesamiento Paralelo por Filas Chen et al [2]

Vishwanath *et al* en [27] presentan dos arquitecturas, la primera es mostrada en la figura 4.2, el diseño se fundamenta en el algoritmo piramidal de Mallat, esencialmente consiste en un módulo de transformación en una dimensión que es utilizado repetidas veces para calcular la 2D-DWT, se requiere de un generador de direcciones para manejar el flujo de los datos y una memoria de almacenamiento de tamaño igual a la imagen de entrada. Se requieren $2N^2$ ciclos de reloj para calcular N filas, por lo tanto se necesitan $4N^2$ ciclos para calcular el primer nivel de la transformación de la imagen. Según las observaciones de los autores la ventaja de la arquitectura es su simplicidad, pero requiere demasiadas celdas de memoria lo que la hace poco conveniente para la implementación en un chip; otra desventaja es la latencia de $2N^2$ ciclos para que el primer dato de salida sea generado.

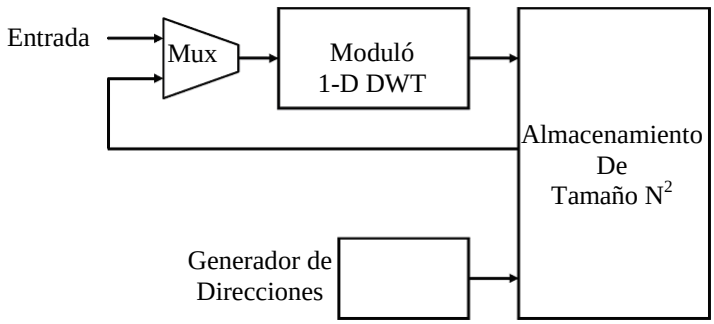


Figura 4. 2 Arquitectura de Aproximación Directa Vishwanath et al [3]

[2] Imagen tomada de:
C. Chen, Z. Yang, T. Wang, and L. Chen, "A programmable VLSI architecture for 2-D discrete wavelet transform", Procedente, IEEE International Symposium on Circuits and Systems, pp. 619-622, Mayo 2000.
[3] Imagen tomada de:
M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," IEEE Transactions on Circuits and Systems - II, vol. 42, no. 3, pp. 305-316, Mayo 1995.

La segunda arquitectura propuesta por Vishwanath *et al* es mostrada en la figura 4.3. Está compuesta por un filtro sistólico (S) que maneja el filtrado en la dirección horizontal, un filtro paralelo (P) para manejar el filtrado en la dirección vertical y una unidad de almacenamiento. Dos filas son procesadas en S1 y S2, los coeficientes calculados son llevados a las celdas de retención las cuales llevan su contenido al bloque de celdas. Las filas se procesan en el orden del esquema del RPA. Los filtros P1 y P2 calculan 4 filas que constituyen cuatro salidas del primer nivel, una de ellas es llevada a S1 para continuar con el procesamiento. Los filtros paralelos producen las salidas en el mismo orden de los filtros de fila, este orden es requerido para la correcta operación de la red de conexión. Esta arquitectura calcula la 2D-DWT de una imagen en $N^2 + N$ ciclos y requiere una memoria de tamaño $2NK$.

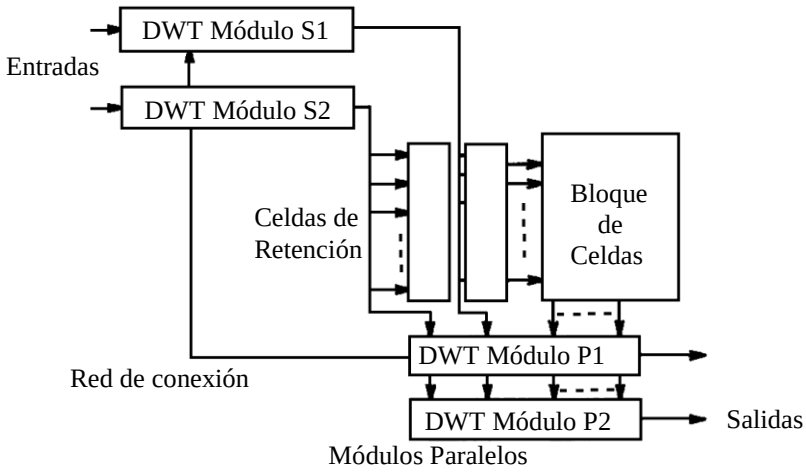


Figura 4. 3 Arquitectura Sistólica-Paralela Vishwanath et al [4]

Colom *et al* en [13] [30] presentan una arquitectura que trabaja con filtros 2D no-separables que utiliza una estructura paralela denominada even-odd. La figura 4.4 muestra el diagrama de bloques de la arquitectura para el cálculo de tres niveles. La arquitectura es recurrente, la unidad de filtrado 1 realiza el cálculo del primer nivel y está continuamente procesando; la unidad de filtrado 2 se encarga del cálculo del resto de niveles, inicia cuando la unidad 1 ha generado las primeras cuatro filas y sigue calculando cada vez que la primera genera dos

[4] Imagen tomada de:
M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," IEEE Transactions on Circuits and Systems - II, vol. 42, no. 3, pp. 305-316, Mayo 1995.

nuevas líneas, los periodos de tiempo intermedios son aprovechados para realizar la recurrencia. Se utilizan dos unidades de almacenamiento que actúan de enlace entre niveles. La arquitectura utiliza unidades de control distribuidas, existe un control al interior de la unidad de filtrado y un control para sincronizar el funcionamiento entre dos niveles consecutivos, estas unidades de control se comunican entre ellas, al incrementar el número de niveles se debe replicar el número de unidades de control. Para el cálculo de una imagen de NxN, se necesita $N^2 + N$ ciclos de reloj y 6N celdas de almacenamiento de memoria.

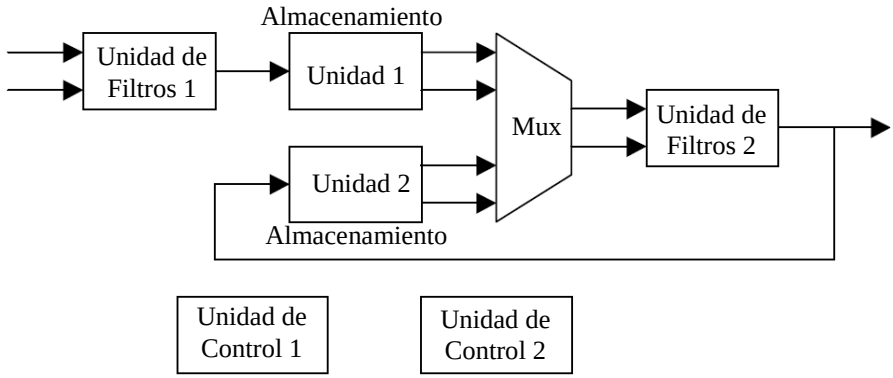


Figura 4. 4 Arquitectura Recurrente para Tres Niveles Colom et al [5]

Sheu *et al* en [31] proponen la arquitectura mostrada en la figura 4.5. La arquitectura consta de dos módulos de filtrado (H1 y H2) para calcular los coeficientes a lo largo de las filas y dos módulos (V1 y V2) para calcular a lo largo de las columnas. Cada módulo se compone de un filtro de paso alto y un filtro de paso bajo. La idea general del funcionamiento es la siguiente: H1 calcula las salidas F_L y F_H, las salidas son almacenadas en la unidad de almacenamiento y después son cargadas en V1 y V2; similarmente en V1 se calcula los coeficientes de F_{LL}, F_{LH} y en V2 se calculan F_{HL}, F_{HH}; la salida de V1 es cargada en H2 donde se calculan F_{LLL}, F_{LLH} y se almacenan los datos para ser pasados a V1 y V2; seguidamente V1 calcula F_{LLLL}, F_{LLLH} y V2 calcula F_{LLHL}, F_{LLHH} y se itera el proceso. Las unidades de filtrado se fundamentan en una metodología de reuso de los datos de entrada con una estructura *paralelo-pipelined*. Para el cálculo de una imagen de NxN, se necesita

[5] Imagen tomada de:
R. J. Colom, R Gadea, A. Sebastiá, M. Martinez, V. Herrero, V. Arnau, "Transformada Discreta Wavelet 2-D para Procesamiento de Vídeo en Tiempo Real," presentado en XII Jornadas de Paralelismo. Valencia, España, 2001.

$N^2 + N$ ciclos de reloj y $N/2 + N/4$ celdas de almacenamiento.

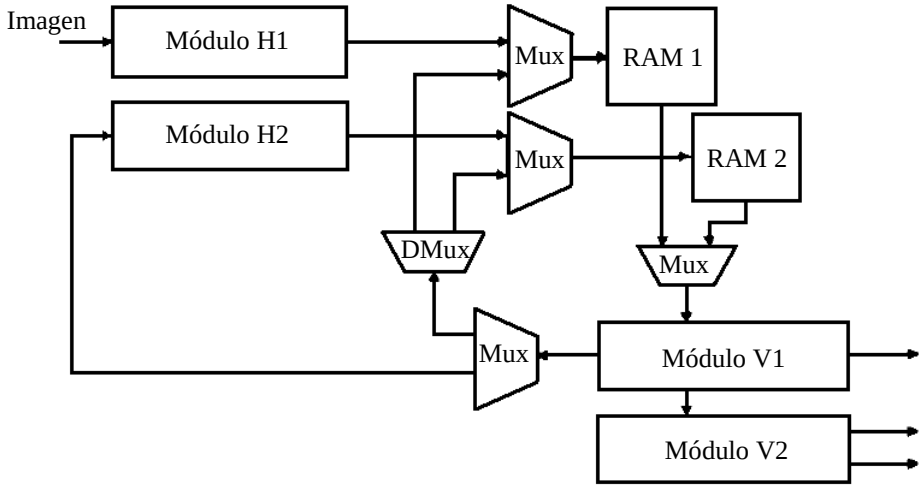


Figura 4. 5 Arquitectura Recurrente para tres Niveles Sheu et al [6]

Chakrabarti *et al* en [32] presentan dos arquitecturas. La primera es mostrada en la figura 4.6. Consta de dos unidades de almacenamiento y cuatro unidades de filtrado paralelas compuestas por un filtro de paso alto y uno de paso bajo. Los filtros 1 y 2 calculan a lo largo de las filas, su salida es almacenada en la unidad 1en filas de mayor orden, los datos de la unidad 1 son leídos por columnas por los filtros 3 y 4 donde se calculan los coeficientes a lo largo de las columnas. Similarmente las salidas de los filtros 3 y 4 son almacenadas en la unidad 2 por columnas y leídas en filas por el filtro 2. En [32] presentan dos algoritmos de planificación del flujo de datos que pueden ser utilizados sobre esta arquitectura. Los autores manifiestan que debido a que las unidades de filtrado son usadas recursivamente para calcular dos subimagenes, se genera un retardo de *N* ciclos lo cual puede ser inaceptable para algunas aplicaciones. Para el cálculo de una imagen se necesita $\approx N^2$ ciclos de reloj y $\approx N/2 + 2KN\left(1 - 1/2^{K}\right) + N\left(1 - 1/2^{K-1}\right)$ celdas de almacenamiento.

[6] Imagen tomada de:
M. Sheu, M. Shieh, S. Liu, "A VLSI Architecture Design With Lower Hardware Cost and Less Memory for separable 2-D Discrete Wavelet Transform," IEEE Circuits and Systems, Procedente de International Symposium on Circuits and Systems, vol. 5, pp. 457-459, Mayo 1998.

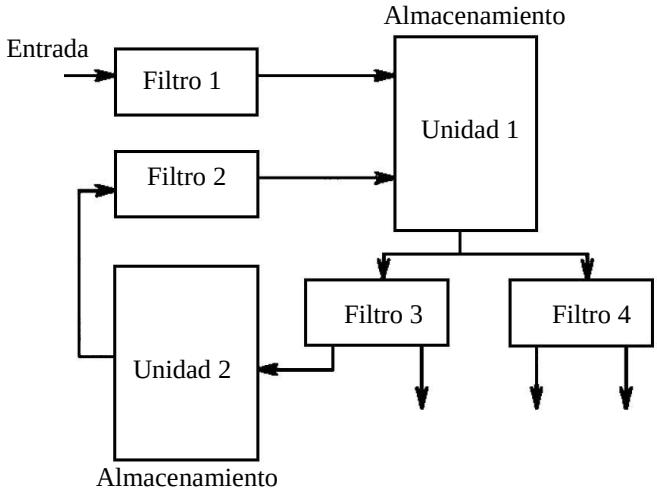


Figura 4. 6 Arquitectura 1 Paralela Chakrabarti et al [7]

La segunda arquitectura propuesta por Chakrabarti *et al* es mostrada en la figura 4.7. Esta arquitectura es una versión modificada de la anterior con el fin de disminuir el retardo generado por la recursividad. En esta propuesta se incrementan dos unidades de filtrado para producir al mismo tiempo la salida de todas las subimagenes de un mismo nivel, con lo cual se logra disminuir el tamaño de las unidades de almacenamiento en $N/2$ y el retardo en *N* ciclos; por consiguiente, para el cálculo de una imagen se necesita $\approx N^2$ ciclos de reloj y $\approx KN\left(1-(1/2)^{K}\right)+N\left(1-(1/2)^{K-1}\right)$ celdas de almacenamiento.

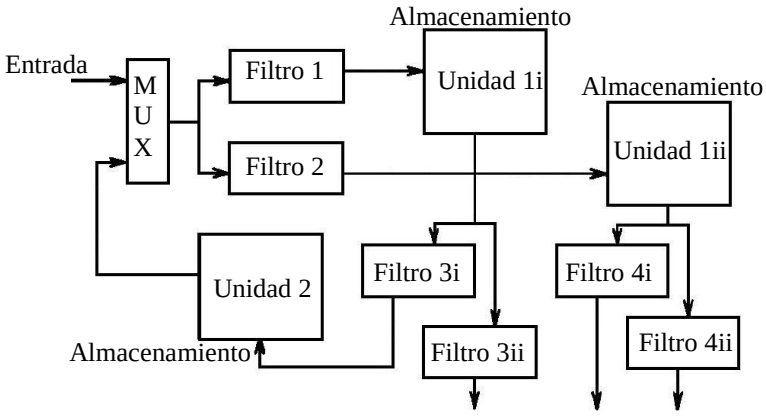


Figura 4. 7 Arquitectura 2 Paralela Chakrabarti et al [8]

[7] Imagen tomada de: *C. Chakrabarti, C. Mumford, "Efficient Realizations of Encoders and Decoder Based on the 2-D Discrete Wavelet Transform," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, no. 3, pp. 289-298, Septiembre 1999.*

[8] Imagen tomada de: *C. Chakrabarti, C. Mumford, "Efficient Realizations of Encoders and Decoder Based on the 2-D Discrete Wavelet Transform," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, no. 3, pp. 289-298, Septiembre 1999.*

4.2.- ARQUITECTURA PROPUESTA

En esta sección se presenta el diseño de la arquitectura propuesta para la implementación de la 2D-DWT, la figura 4.8 muestra el diagrama de bloques.

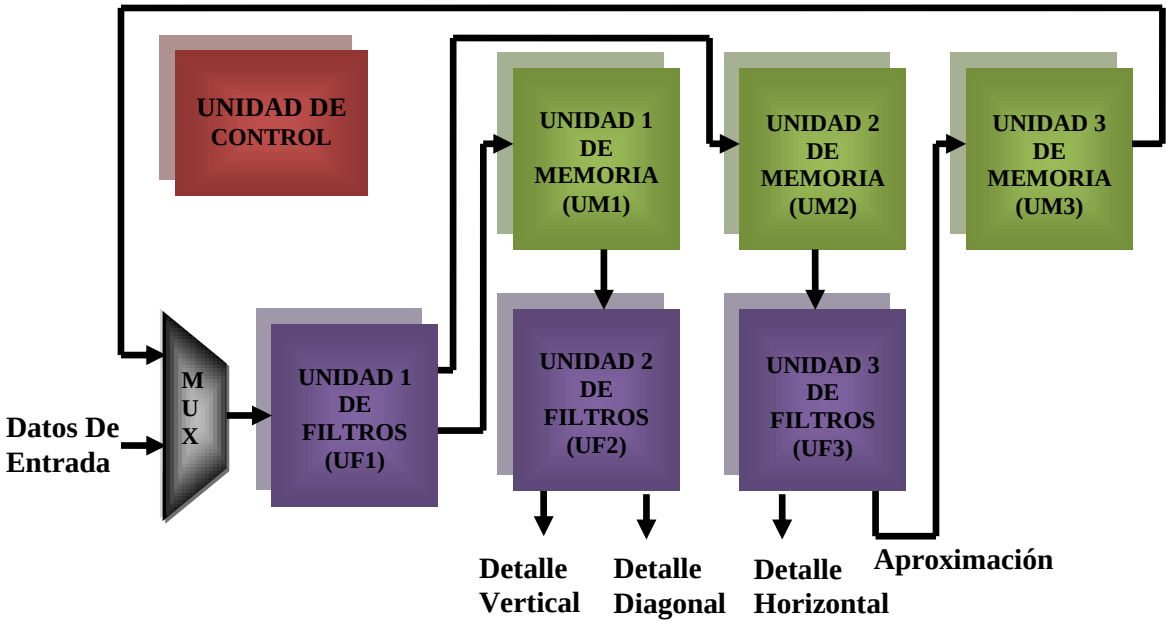


Figura 4. 8 Arquitectura Paralela Por Nivel

La arquitectura está compuesta por tres unidades de almacenamiento, una unidad de control y tres unidades de filtros paralelas cada una compuestas por un filtro de paso alto y uno de paso bajo. La unidad de filtros 1 calcula a lo largo de las filas, su salida es almacenada en las unidades de memoria 1 y 2, en la unidad de memoria 1 se almacenan los coeficientes de la banda alta y en la unidad de memoria 2 los de la banda baja. Las unidades de filtrado 2 y 3 calculan a lo largo de las columnas, los coeficientes de la banda baja de la unidad de filtros 3, se almacenan en la unidad de memoria 3, para posteriormente pasar a la unidad de filtros 1 para el cálculo del siguiente nivel.

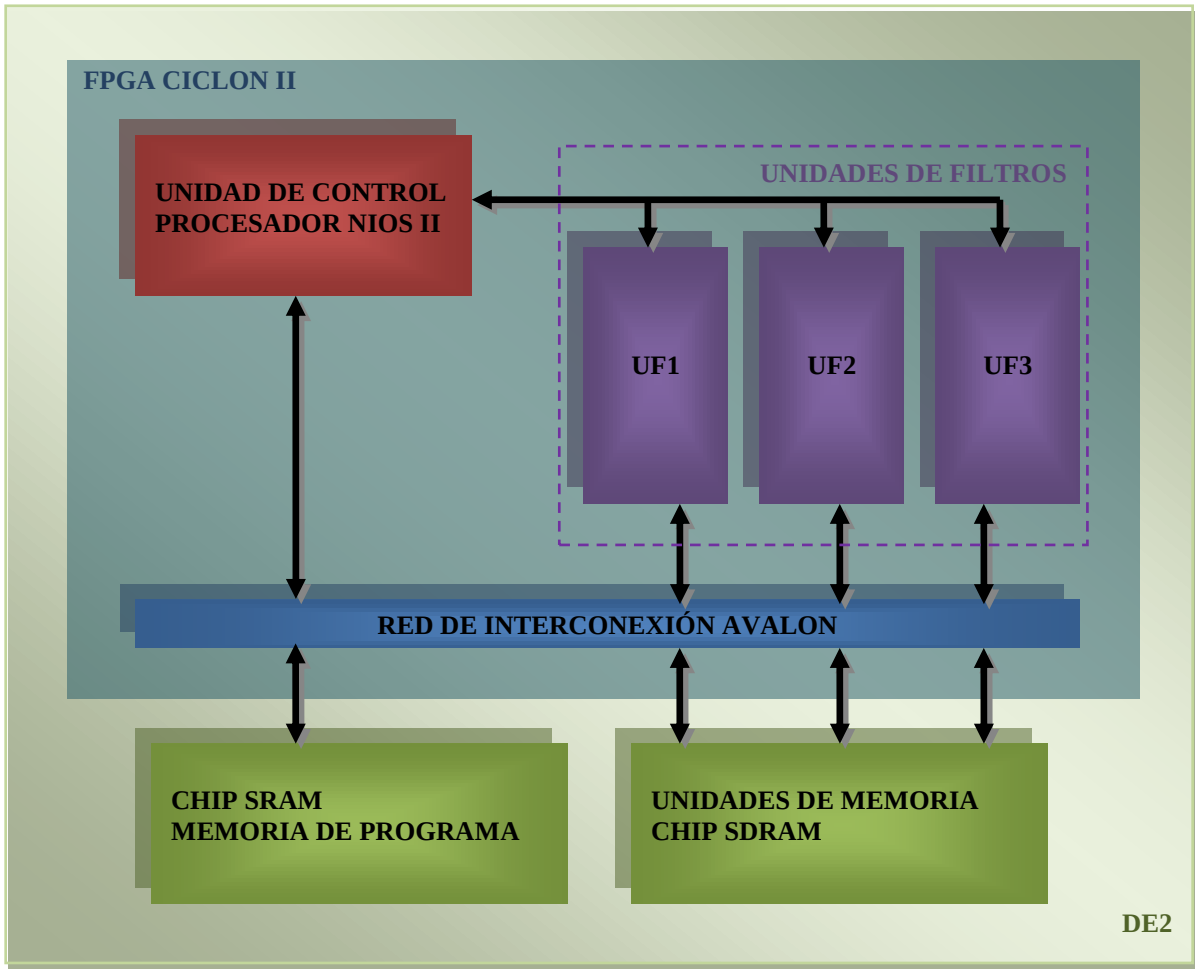


Figura 4. 9 Diagrama de Bloques del Sistema Implementado en la Tarjeta de Desarrollo DE2

La unidad de control se encarga de la planificación del flujo de datos de la siguiente manera: UF1 calcula las salidas F_H y F_L (ver figura 2.7) , las salidas son almacenadas en la UM1 y UM2, después son cargadas en UF2 y UF3; de igual forma en UF2 se calculan F_{HL}, F_{HH} y en UF3 se calcula los coeficientes de F_{LL}, F_{LH} ; la salida de la componente baja de UF3 es almacenada en UM3 para posteriormente ser cargada en UF1 donde se calculan F_{LLL}, F_{LLH} y se itera el proceso.

El diseño es implementado sobre la tarjeta de desarrollo DE2 [35], esta tarjeta cuenta con una FPGA Ciclone II EP2C35F672C6 [33] y con unidades de memoria e interfaces de entrada y salida que la convierten en una plataforma ideal para prototipado de aplicaciones

en multimedia y redes. La figura 4.9 muestra el diagrama de bloques del sistema implementado sobre la DE2.

4.2.1- Unidad de Memoria

Como se menciona en la sección anterior las unidades de memoria son utilizadas para almacenar los datos resultado de la transformación; la UM1 y la UM2 almacenan los resultados de la transformación a lo largo de las filas (F_H, F_L) para lo cual requieren de *NxN/2* celdas de almacenamiento, la UM3 almacena los resultados de la componente baja frecuencia de la transformación por columnas F_{LL} para lo cual requiere de *N/2xN/2* celdas, por lo tanto el total de memoria requerida por la arquitectura propuesta está determinado por:

$$T_{MEM} = N.\frac{N}{2} + N.\frac{N}{2} + \frac{N}{2}.\frac{N}{2} = \frac{5}{4}N^2$$

La FPGA Ciclone II EP2C35F672C6 cuenta con una estructura de memoria interna organizada en 3 columnas que contienen un total de 105 bloques que proveen una capacidad de almacenamiento al interior de la FPGA de 483840bits y una velocidad máxima de operación de 250MHz [33], lo cual solo permitiría procesar imágenes con N inferior a 220 píxeles; aunque se puede ampliar la capacidad de almacenamiento interno utilizando los bloques de arreglos lógicos para almacenar datos, no es recomendable puesto que solo se aumentan 2047bytes utilizando todos los recursos de la FPGA, la figura 4.10 muestra la relación entre el tamaño de la memoria y el consumo de elementos lógicos, registros y bloques de memoria; se nota que una vez se alcanza el límite de almacenamiento de los bloques de memoria se eleva el consumo de elementos lógicos y registros. Para obtener los datos de esta gráfica se modelaron unidades de memoria de diferentes tamaños utilizando VHDL (very-high-speed integrated circuits hardware description language en español lenguaje de descripción de hardware de circuitos integrados de muy alta velocidad) y las megafunciones de altera, el código esté disponible en el Anexo B.

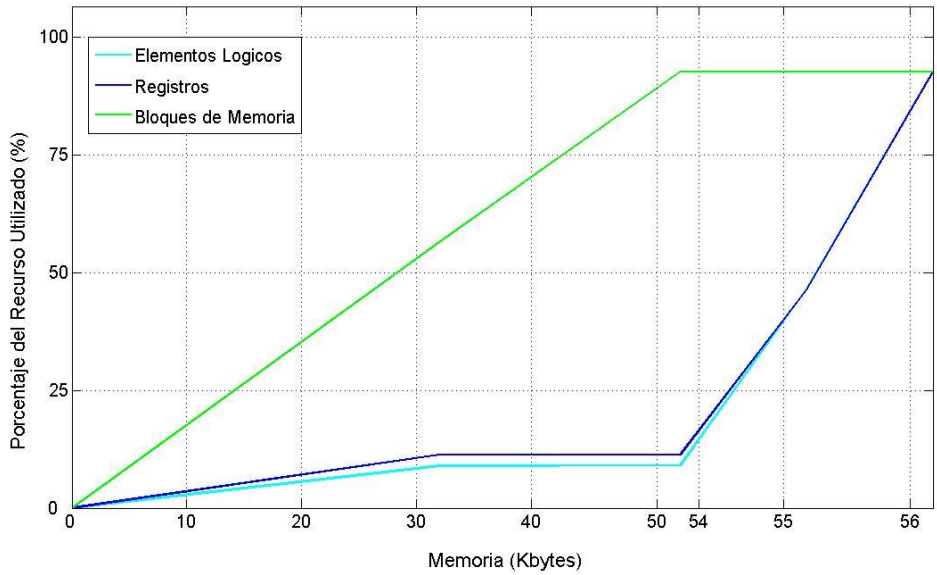


Figura 4. 10 Relación Entre el Tamaño de la Memoria y el Consumo de Elementos Lógicos y Registros

Para procesar imágenes de mayor tamaño se amplio la capacidad de almacenamiento del sistema utilizando una de las memorias de la tarjeta DE2, se uso la SDRAM IS42S16400 que almacena 8Mbytes de datos, para la interfaz de conexión Altera ha desarrollado una herramienta llamada SOPC (System on a Programable Chip) [36] que permite manipular controladores predefinidos y recurre a la red de interconexión AVALON que maneja menos elementos lógicos en la conexión y mejora el desempeño en la velocidad de transmisión. La figura 4.9 muestra el diagrama de bloques de la arquitectura propuesta implementada sobre la DE2, se observa que para la implementación de las unidades de memoria se recurre a una SDRAM externa a la FPGA.

4.2.2- Unidad de Filtros

La unidad de filtros se fundamentan en una metodología de reuso de los datos de entrada con una estructura paralelo-pipelined similar a la propuesta por Sheu *et al* en [31] pero el cálculo de los coeficientes se realiza con la metodología even-odd propuesta por Colom *et al* en [13] [30], con lo que se logra calcular un coeficiente por ciclo después de 2 ciclos de

latencia, el esquema general de la unidad de filtros parte baja se presenta en la figura 4.11. Esta unidad está compuesta por un registro de corrimiento que almacenan los coeficientes del filtro y está configurado de manera que en cada ciclo realiza dos desplazamientos. Existen tres unidades MSA (Multiplicador-Sumador-Acumulador) compuestas por dos multiplicadores (uno para los datos pares y otro para los datos impares) un sumador y un registro acumulador, finalmente se dispone un multiplexor que selecciona cuál de los datos se encuentra listo para ser enviado a la siguiente etapa.

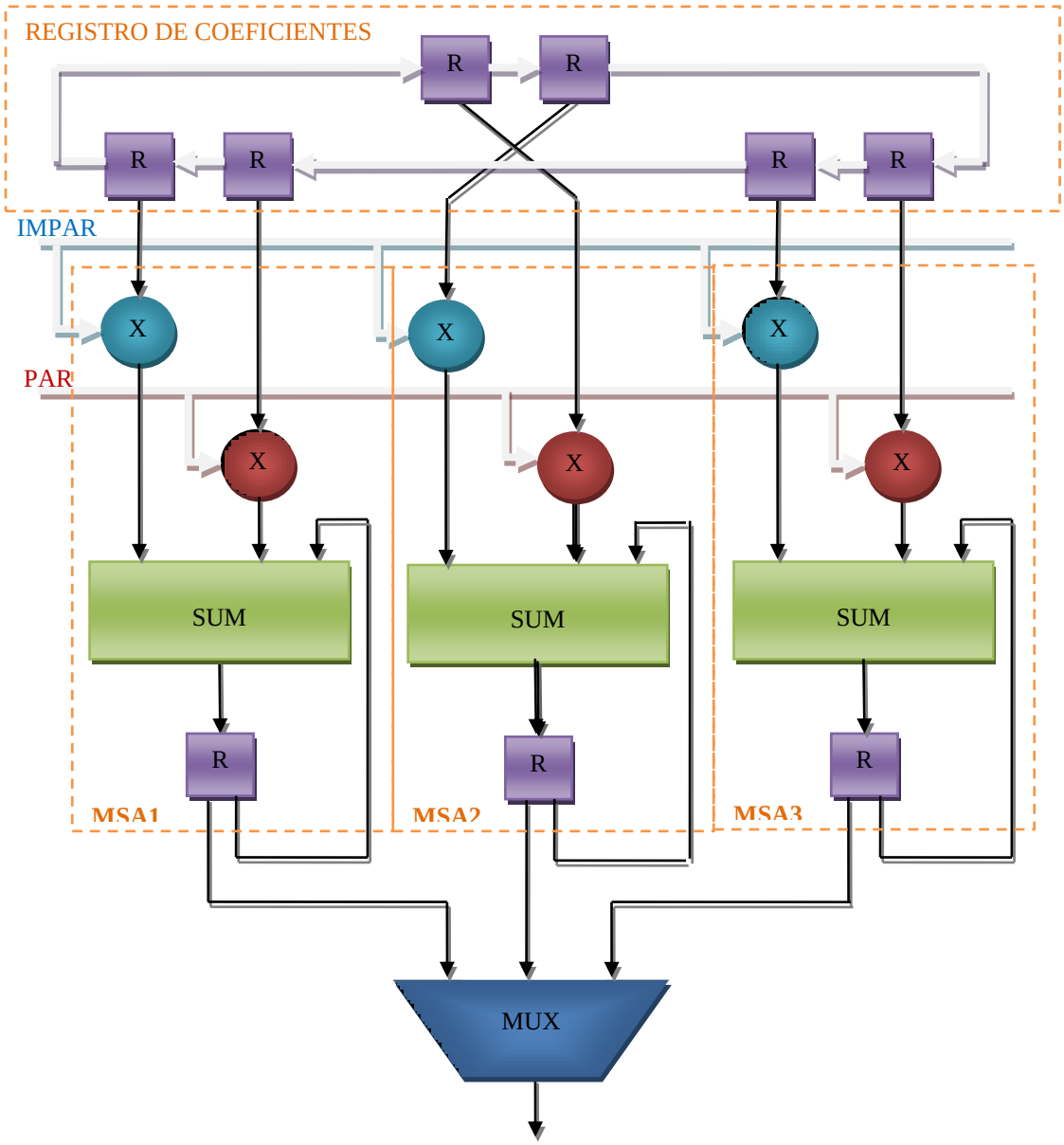


Figura 4. 11 Estructura Base de la Unidad de Filtros Parte Baja

Para ilustrar el funcionamiento de esta unidad se consideraron dos filtros H (paso bajo) y G (paso alto) con seis coeficientes definidos de la siguiente manera:

$$H = [h_1, h_2, h_3, h_4, h_5, h_6]$$

$$G = [g_1, g_2, g_3, g_4, g_5, g_6]$$

Sea F una señal con N datos, cuyos valores discretos están definidos así:

$$F = [f_1, f_2, f_3, f_4, \ldots f_N]$$

Sean A y D las señales de salida cuyos valores discretos corresponden a los componentes de baja y alta frecuencia, resultado de la transformación:

$$A = \left[A_1, A_2, A_3, A_4, \ldots A_{N/2} \right]$$

$$D = \left[D_1, D_2, D_3, D_4, \ldots D_{N/2} \right]$$

Se explicará el funcionamiento de esta unidad utilizando los coeficientes de paso bajo de la transformación en una dimensión, de acuerdo a la ecuación 2.8 (presentada previamente en el capítulo 2):

$$A_1 = f_1.h_1 + f_2.h_2 + f_3.h_3 + f_4.h_4 + f_5.h_5 + f_6.h_6$$

$$A_2 = f_3.h_1 + f_4.h_2 + f_5.h_3 + f_6.h_4 + f_7.h_5 + f_8.h_6$$

$$A_3 = f_5.h_1 + f_6.h_2 + f_7.h_3 + f_8.h_4 + f_9.h_5 + f_{10}.h_6$$

$$A_4 = f_7.h_1 + f_8.h_2 + f_9.h_3 + f_{10}.h_4 + f_{11}.h_5 + f_{12}.h_6$$

Hasta llegar a los límites de la señal donde:

$$A_{N/2} = f_{N-1}.h_1 + f_N.h_2$$

La metodología de reuso consiste en organizar el flujo de los datos de entrada de forma que sea posible calcular en paralelo varios de los datos de salida así:

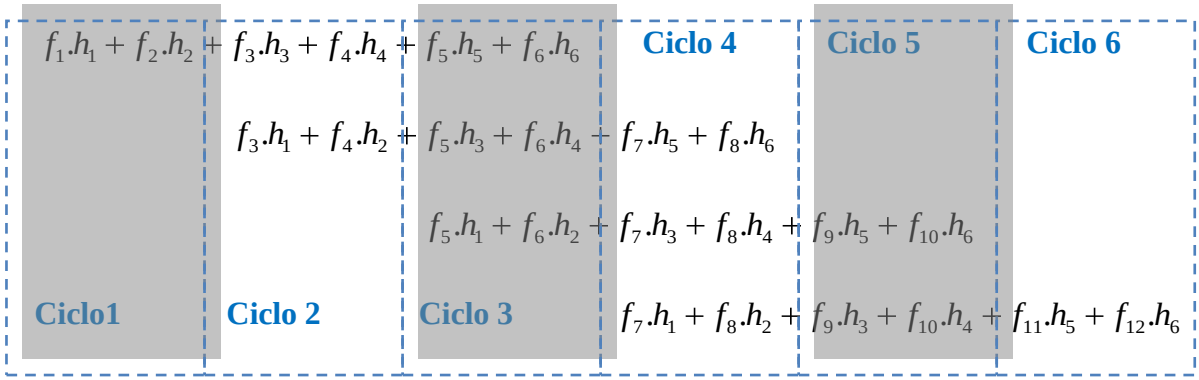


Figura 4. 12 Flujo de Datos por Ciclo

Como se mencionó anteriormente en cada ciclo el registro de coeficientes se desplaza dos espacios; así en el primer ciclo a MSA1 llegan los coeficientes h_1, h_2, a MSA2 llegan h_3, h_4 y a MSA3 llegan h_5, h_6; en el segundo ciclo a MSA1 llegan los coeficientes h_3, h_4, a MSA2 llegan h_5, h_6 y a MSA3 llegan h_1, h_2; en el tercer ciclo a MSA1 llegan los coeficientes h_5, h_6, a MSA2 llegan h_1, h_2 y a MSA3 llegan h_3, h_4 y el proceso se repite hasta finalizar la transformación. Por otro lado en cada ciclo entran de manera paralela dos datos (par e impar) para ser procesados; en el primer ciclo ingresan f_1, f_2 con estos datos el MSA1 inicia el cálculo de A_1 acumulando $f_1.h_1 + f_2.h_2$; en el segundo ciclo ingresan f_3, f_4, el MSA1 acumula $f_3.h_3 + f_4.h_4$ y de manera paralela MSA2 utiliza los mismos datos de entrada para iniciar el cálculo de A_2 acumulando $f_3.h_1 + f_4.h_2$; en el tercer ciclo ingresan f_5, f_6, el MSA1 calcula $f_5.h_5 + f_6.h_6$ con lo que finaliza el cálculo de A_1, de manera paralela MSA2 calcula $f_5.h_3 + f_6.h_4$ y MSA3 inicia el cálculo de A_3 acumulando $f_5.h_1 + f_6.h_2$, en adelante se entregará un dato de salida en cada ciclo y el multiplexor conmuta para transportarlo a la salida; en el cuarto ciclo ingresan f_7, f_8, el MSA1 inicia el cálculo de A_4 acumulando $f_7.h_1 + f_8.h_2$, el MSA2 calcula $f_7.h_5 + f_8.h_6$ con lo que finaliza el cálculo de A_2, el MSA3 acumula $f_7.h_3 + f_8.h_4$; en el quinto ciclo ingresan f_9, f_{10}, el MSA1 acumula $f_9.h_3 + f_{10}.h_4$, el MSA2 inicia el cálculo de A_5 y el MSA3 finaliza el

cálculo A_3 acumulando $f_9.h_5 + f_{10}.h_6$; en adelante el proceso se itera hasta llegar al final de la señal de entrada.

Siguiendo la metodología descrita en el párrafo anterior se calculan los componentes de paso alto de la señal de salida en una arquitectura paralela, la única diferencia se encuentra en los datos almacenados en el registro de corrimiento que deben corresponder a los coeficientes del filtro de paso alto. Por lo tanto la estructura completa de la unidad de filtros se muestra en la figura 4.13.

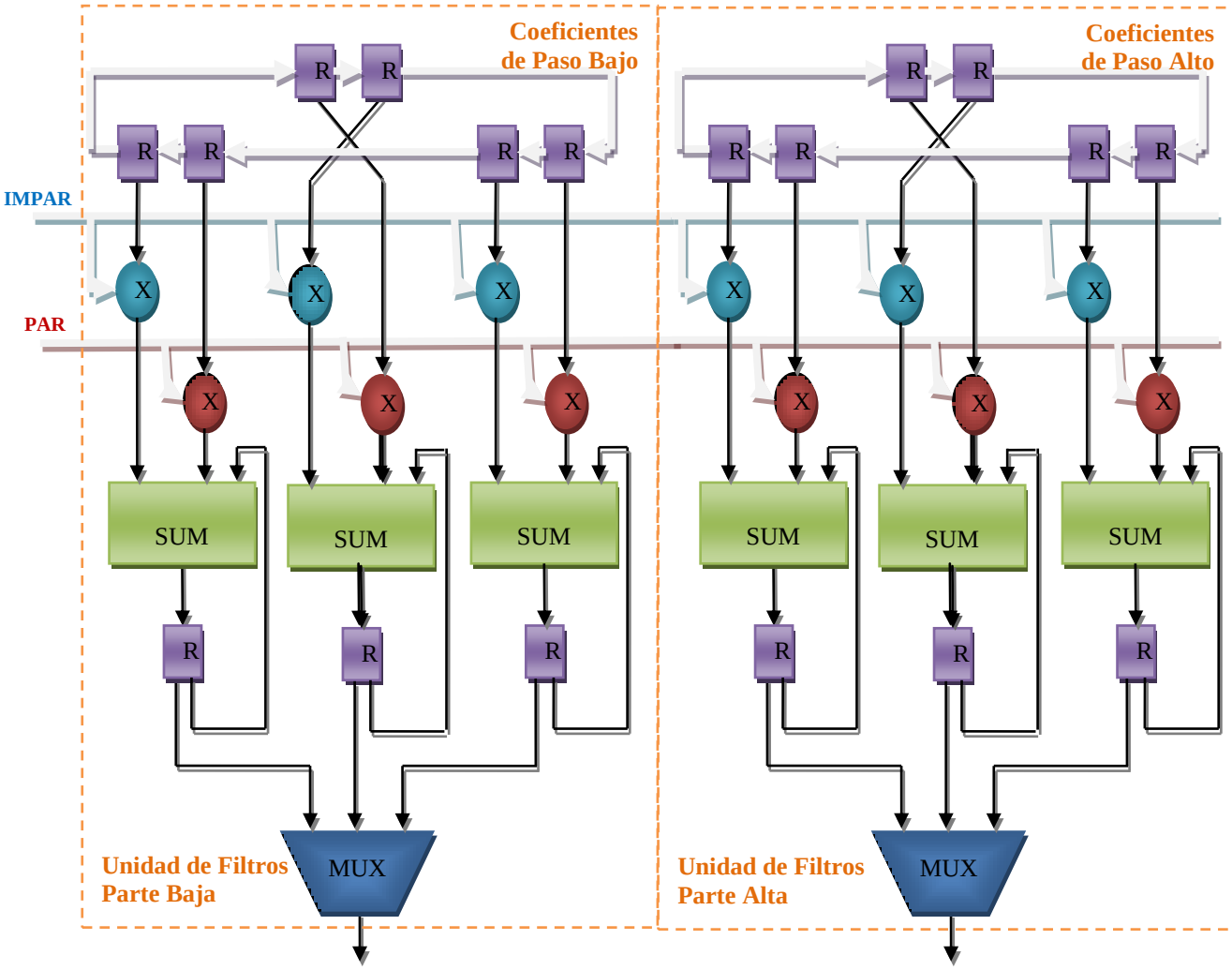


Figura 4. 13 Unidad de Filtros

Note que cada dato de entrada se utiliza en el cálculo parcial de varios de los coeficientes de salida. Al realizar el procesamiento de los pares en paralelo con los impares se obtienen dos

datos de salida (uno de paso alto y uno de paso bajo) en cada ciclo de reloj a partir del tercer ciclo. Por lo tanto la transformación de una señal unidimensional de tamaño *N* se realiza en $\frac{N}{2}+2$ ciclos de reloj, si ampliamos el procesamiento para una señal en dos dimensiones de tamaño *NxN* se requieren $\left(\frac{N}{2}+2\right)N$ ciclos para procesar todas las filas y $\left(\frac{N}{2}+2\right)\frac{N}{2}$ ciclos para procesar las columnas, por lo tanto el total de ciclos requeridos por la arquitectura propuesta para calcular los coeficientes del primer nivel de transformación está determinado por:

$$T_{CICLOS} = \left(\frac{N}{2}+2\right)N + \left(\frac{N}{2}+2\right)\frac{N}{2} = \frac{3}{4}N^2 + 3N$$

El algoritmo de planificación del flujo al interior de la unidad de filtros puede expresarse de la siguiente manera:

X=1
PARA i=1 HASTA N CON INCREMENTO +2
$A_X = f_i h_1 + f_{i+1} h_2$
$D_X = f_i g_1 + f_{i+1} g_2$
SI (x>1) ENTONCES
$A_{X-1} = A_{X-1} + f_i h_3 + f_{i+1} h_4$
$D_{X-1} = D_{X-1} + f_i g_3 + f_{i+1} g_4$
FIN SI
SI (x>2) ENTONCES
$A_{X-2} = A_{X-2} + f_i h_5 + f_{i+1} h_6$
$D_{X-2} = D_{X-2} + f_i g_5 + f_{i+1} g_6$
FIN SI
X=X+1
FIN PARA

Esta unidad se modeló utilizando VHDL y las megafunciones de Altera, el diseño tiene 5 módulos estructurales: Registro de Coeficientes, Multiplicador, Sumador, Registro Acumulador y Multiplexor. Para la implementación de los multiplicadores se utilizó la megafunción de Altera *lpm_mul* que permite utilizar los multiplicadores embebidos optimizando el uso de la FPGA, para el Multiplexor se utilizó la megafunción de Altera

lpm_mux dado que al sintetizar el diseño descrito en VHDL se requerían 17 elementos lógicos adicionales; el módulo sumador se implementó con la megafunción *parallel_add* puesto que el diseño descrito en VHDL requería de un tiempo de estabilización mayor en 0.577ns, los elementos restantes fueron modelados en VHDL. El código en VHDL y la descripción de estos módulos se encuentran en el Anexo B.

Tabla 4. 1 Recursos Utilizados en la Unidad de Filtros

Tipo	**Recursos Utilizados**
Elementos Lógicos	292/33216 (<1%)
Registros	219
Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	24/70 (34%)
PLL	0/4 (0%)

La unidad de filtros fue sintetizada en una FPGA Ciclone II EP2C35F672C6 [33] de Altera utilizando la herramienta de diseño Quartus II versión 6.0 web edition, la tabla 4.1 muestra los recursos usados en la implementación de la unidad de filtros y la tabla 4.2 muestra la jerarquía de los módulos y los recursos por módulo.

De las tablas 4.1 y 4.2 observamos que en la implementación de la unidad de filtros se utiliza el 34% de los multiplicadores embebidos, el 1% de los elementos lógicos y 0% de los bits de memoria, ello evidencia que disponemos de suficiente espacio al interior de FPGA para aumentar el grado de paralelismo en el diseño y utilizar simultáneamente varias unidades de filtros, lo que permitiría obtener los coeficientes de transformación en menor tiempo pero ocupando la totalidad de los multiplicadores disponibles, en aplicaciones donde se utilice la FPGA sólo para la implementación de la etapa de transformación ésta es una opción viable, sin embargo en nuestra aplicación se propone la implementación de todo el sistema de compresión en la FPGA y por lo tanto se plantea el uso recursivo de las unidades de filtros para el cálculo de los siguientes niveles de transformación.

Tabla 4. 2 Jerarquía de los Módulos y Recursos Utilizados en la Unidad de Filtros

Entity	Logic Cells	LC Combinationals	LC Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C35F672C6											
DWT	292 (0)	270 (0)	219 (0)	0	0	24	0	12	73 (0)	22 (0)	197 (0)
UF :inst	292 (0)	270 (0)	219 (0)	0	0	24	0	12	73 (0)	22 (0)	197 (0)
lpm_mult0:\ConexMultG:1:Mult1G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult:lpm_mult_component	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
mult_2sr:auto_generated	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultG:1:Mult2G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultG:2:Mult1G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultG:2:Mult2G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultG:3:Mult1G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultG:3:Mult2G	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:1:Mult1H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:1:Mult2H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:2:Mult1H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:2:Mult2H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:3:Mult1H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
lpm_mult0:\ConexMultH:3:Mult2H	0 (0)	0 (0)	0 (0)	0	0	2	0	1	0 (0)	0 (0)	0 (0)
Registro:\ConexRegG:1:RegG	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	0 (0)	32 (32)
Registro:\ConexRegG:2:RegG	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	2 (2)	30 (30)
Registro:\ConexRegG:3:RegG	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	0 (0)	32 (32)
Registro:\ConexRegH:1:RegH	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	6 (6)	26 (26)
Registro:\ConexRegH:2:RegH	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	1 (1)	31 (31)
Registro:\ConexRegH:3:RegH	32 (32)	0 (0)	32 (32)	0	0	0	0	0	0 (0)	0 (0)	32 (32)
parallel_add0:\ConexSumG:1:SumG	32 (0)	32 (0)	0 (0)	0	0	0	0	0	7 (0)	0 (0)	25 (0)
parallel_add:parallel_add_component	32 (0)	32 (0)	0 (0)	0	0	0	0	0	7 (0)	0 (0)	25 (0)
par_add_rje:auto_generated	32 (32)	32 (32)	0 (0)	0	0	0	0	0	7 (7)	0 (0)	25 (25)
parallel_add0:\ConexSumG:2:SumG	32 (0)	32 (0)	0 (0)	0	0	0	0	0	7 (0)	0 (0)	25 (0)
parallel_add0:\ConexSumG:3:SumG	32 (0)	32 (0)	0 (0)	0	0	0	0	0	9 (0)	0 (0)	23 (0)
parallel_add0:\ConexSumH:1:SumH	32 (0)	32 (0)	0 (0)	0	0	0	0	0	11 (0)	0 (0)	21 (0)
parallel_add0:\ConexSumH:2:SumH	32 (0)	32 (0)	0 (0)	0	0	0	0	0	8 (0)	0 (0)	24 (0)
parallel_add0:\ConexSumH:3:SumH	32 (0)	32 (0)	0 (0)	0	0	0	0	0	5 (0)	0 (0)	27 (0)
lpm_mux0:ConexMuxG	32 (0)	32 (0)	0 (0)	0	0	0	0	0	11 (0)	0 (0)	21 (0)
lpm_mux0:ConexMuxH	32 (0)	32 (0)	0 (0)	0	0	0	0	0	15 (0)	0 (0)	17 (0)
RegCoeficientesG:ConexRegCoefG	12 (12)	8 (8)	12 (12)	0	0	0	0	0	0 (0)	4 (4)	8 (8)
RegCoeficientesH:ConexRegCoefH	15 (15)	6 (6)	15 (15)	0	0	0	0	0	0 (0)	9 (9)	6 (6)

En la figura 4.14 se muestran los resultados de la simulación de la unidad de filtros, en el primer ciclo de reloj se realiza la carga de los registros de coeficientes con los valores de los filtros utilizando la señal de control de carga (Load) y se limpian los registros acumuladores usando las señal de limpieza (Clear) , en el segundo ciclo se realizan operaciones parciales; a partir del tercer ciclo se obtienen dos coeficientes de salida en cada ciclo para lo cual se conmutan los multiplexores a través de su señal de selección (Sel), es necesario cada tres ciclos generar señales de limpieza escaladas para borrar los registros acumuladores, los datos pares están representados por la señal llamada Fpar y los impares por Fimpar, los coeficientes de alta frecuencia de la transformación por la señal de salida A y los de baja frecuencia por D; se validaron los valores obtenidos utilizando un programa en Matlab (disponible en el Anexo A) los resultados se tabulan en la tabla 4.3, donde se observa que los datos obtenidos en la transformación concuerdan totalmente (error 0%), lo cual se ajusta a lo esperado puesto que en esta etapa los datos aun no han sido cuantificados

ni comprimidos y es en estas etapas donde se presenta una disminución en la precisión.

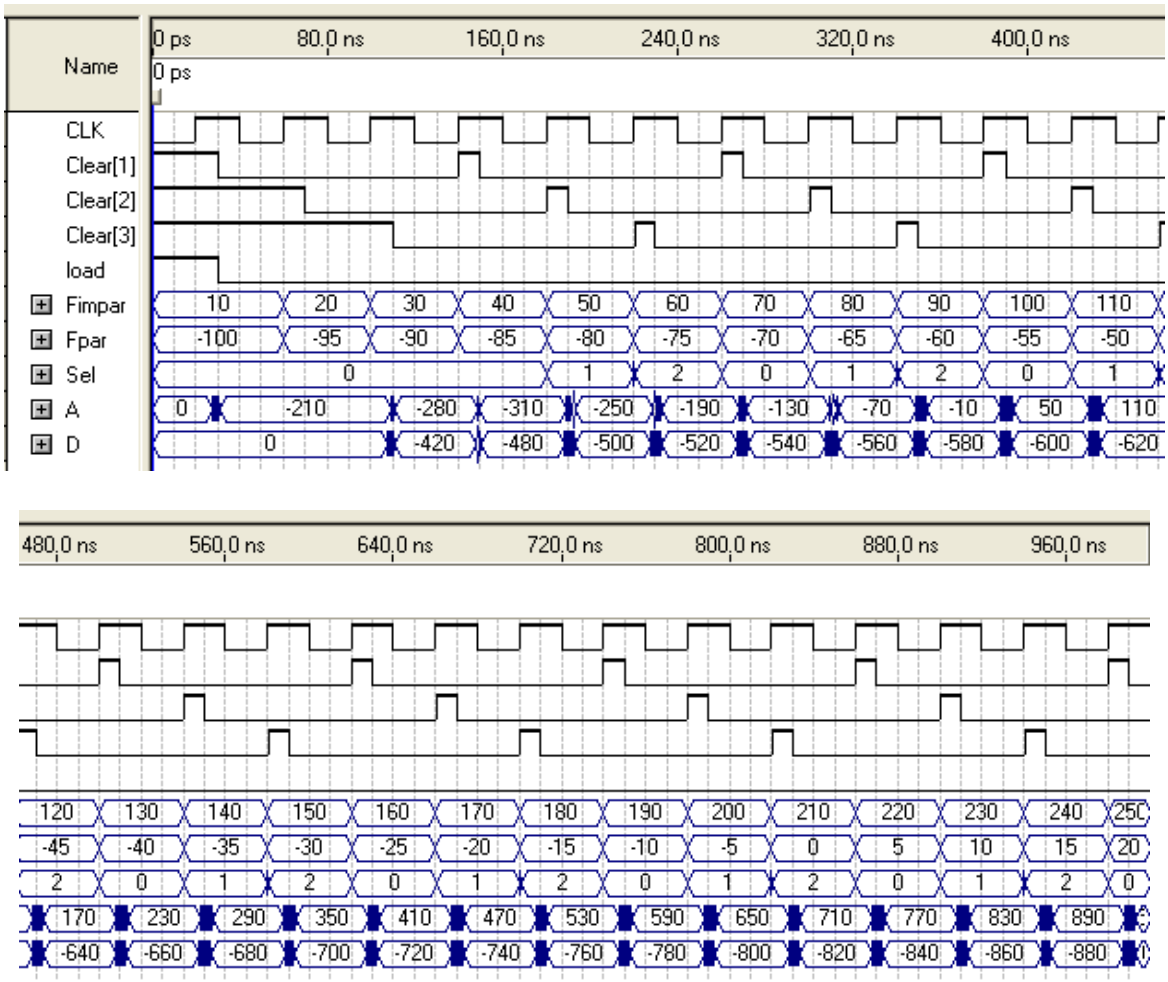


Figura 4. 14 Simulación Unidad de Filtros

Tabla 4. 3 Validación de la Simulación de la Unidad de Filtros

SALIDA SIMULADA EN QUARTUS		SALIDA PROCESADA EN MATLAB	
A	**D**	**A**	**D**
-310	-480	-310	-480
-250	-500	-250	-500
-190	-520	-190	-520
-130	-540	-130	-540
-70	-560	-70	-560

-10	-580	-10	-580
50	-600	50	-600
110	-620	110	-620
170	-640	170	-640
230	-660	230	-660
290	-680	290	-680
350	-700	350	-700
410	-720	410	-720
470	-740	470	-740
530	-760	530	-760
590	-780	590	-780
650	-800	650	-800
710	-820	710	-820
770	-840	770	-840
830	-860	830	-860
890	-880	890	-880
950	-900	950	-900
1250	-440	1250	-440
-370	-460	-370	-460

4.2.2- Unidad de Control

El control de la arquitectura se realiza utilizando un diseño soportado en el sistema Nios II [37], Nios II es un procesador definido en un lenguaje de descripción hardware que puede ser implementado en FPGAs de Altera usando la herramienta de síntesis Quartus II en conjunción con el SOPC Builder. Existen tres versiones del procesador: la económica Nios/e, la estándar Nios/s y la rápida Nios/f. La versión económica tiene el núcleo de menor tamaño pero el desempeño es el más bajo y no maneja memoria cache ni hardware

especializado para desarrollar operaciones aritméticas. La versión rápida es la de mayor velocidad de procesamiento pero su núcleo tiene el mayor tamaño, maneja cache para datos e instrucciones, maneja hardware especializado para operaciones aritméticas y también permite la ejecución en modo de supervisión. La versión estándar provee un diseño que se encuentra balanceado en tamaño y velocidad, maneja cache para instrucciones pero no para datos y maneja hardware especializado para operaciones aritméticas [37]. La selección de la versión se realiza de acuerdo a las necesidades del sistema, para el control de la arquitectura se escogió la versión económica debido a que en la unidad de control no realiza operaciones complejas, esta unidad usa señales sencillas para direccionar los datos y sincronizar las unidades de filtros con las de memoria.

Una vez se ha configurado el hardware del sistema Nios en la FPGA es necesario crear un programa que permita realizar las funciones requeridas, el programa puede escribirse en lenguaje ensamblador o en lenguaje C; Altera posee un entorno de desarrollo integrado NIOS II IDE que permite compilar, depurar y cargar fuentes en C/C++ sobre el sistema desarrollado; también es preciso destinar un espacio en memoria para cargar el programa a ejecutar, la tarjeta DE2 posee un chip de memoria SRAM que almacena 512Kbytes y se incluye en el sistema como memoria de programa, al igual que la SDRAM la SRAM se conecta a la red AVALON utilizando el SOPC Builder.

El programa de la unidad de control se desarrolla en C++, el código fuente puede ser consultado en el anexo C, el diagrama de flujo del programa principal se presenta en la figura 4.15 y el diagrama de flujo de la función para el cálculo de la transformada en una dimensión se presenta en la figura 4.16.

4.2.3- Comparación de las Características Entre las Arquitecturas Para el Cálculo de la Transformada Wavelet

El diseño de una arquitectura debe estar en concordancia con los requerimientos del sistema para el cual fue desarrollada; en el caso de las arquitecturas para el cálculo de la 2D-DWT implementadas sobre FPGAs se busca primordialmente mejorar características como: el

tiempo de procesamiento, los requerimientos de memoria, la complejidad en el ruteo y el consumo de recursos. La tabla 4.4 relaciona estas características para las arquitecturas consultadas y la arquitectura propuesta en este trabajo.

Teniendo en cuenta que las FPGAs contienen un número limitado de celdas lógicas, es necesario que el diseño de las arquitecturas se oriente hacia la optimización de este recurso; se requiere un alto número de celdas lógicas para la implementación de elementos de almacenamiento, por lo tanto es conveniente utilizar arquitecturas con un menor requerimiento de memoria; la complejidad del ruteo también lleva a alto consumo de celdas lógicas, pues se requieren más elementos para interconectar el sistema, siendo conveniente una red de ruteo poco compleja; otro factor primordial es el tiempo necesario para el cálculo de la transformación, éste debe reducirse al mínimo para procesar grandes cantidades de imágenes a tasas satisfactorias; de igual forma es conveniente reducir la complejidad del sistema de control para que la arquitectura sea fácilmente escalable y programable. El logro de todos estos objetivos es particularmente difícil en un FPGA ya que área y velocidad son inversamente proporcionales y la satisfacción de un requerimiento significa afectar de manera sensible el otro, por ejemplo, se puede aumentar la velocidad con un alto grado de paralelismo lo que implica un costo en el área. Es importante alcanzar un equilibrio entre estas características obteniendo velocidades aceptables dentro de los límites del área delimitada por el dispositivo a utilizar.

Tabla 4. 4 Principales Características de las Arquitecturas del Estado del Arte

Arquitectura	**Requerimiento de Almacenamiento**	**Tiempo de Calculo**	**Ruteo**	**Control**
Arquitectura Propuesta en Este Trabajo	$\frac{5}{4}N^2$	$\frac{3}{4}N^2+3N$	simple	moderado
Arquitectura de Procesamiento Paralelo por Filas Chen et al	$(K+1)NJ$	N^2+N	complejo	complejo
Arquitectura de Aproximación	N^2	$4N^2$	simple	simple

Directa Vishwanath et al				
Arquitectura Sistólica-Paralela Vishwanath et al	$2NK$	$N^2 + N$	complejo	complejo
Arquitectura Recurrente para Tres Niveles Colom et al	$6N$	$N^2 + N$	moderada	moderada
Arquitectura Recurrente para Tres Niveles Sheu et al	$N/2 + N/4$	$N^2 + N$	moderada	moderada
Arquitectura 1 Paralela Chakrabarti et al	$\approx N(3/2 - 2^{1-J}) + KN(2 - 2^{1-J})$	$\approx N^2$	moderada	moderada
Arquitectura 2 Paralela Chakrabarti et al	$\approx KN(2 - 2^{-J}) + N(2 - 2^{1-J})$	$\approx N^2$	moderada	moderada

De la tabla 4.4 podemos observar que la arquitectura propuesta en este trabajo presenta una disminución en el tiempo de cálculo en comparación con las arquitecturas consultadas, en el caso mas notable es de aproximadamente $3N^2$ ciclos y en el caso menos notable es de aproximadamente $0.25\ N^2$ ciclos, la mejora no complica la implementación pues se conserva un ruteo simple y una unidad de control de complejidad moderada. Si bien en nuestra propuesta se requieren más celdas de almacenamiento este percance se supera al utilizar una memoria externa para el almacenamiento de los datos.

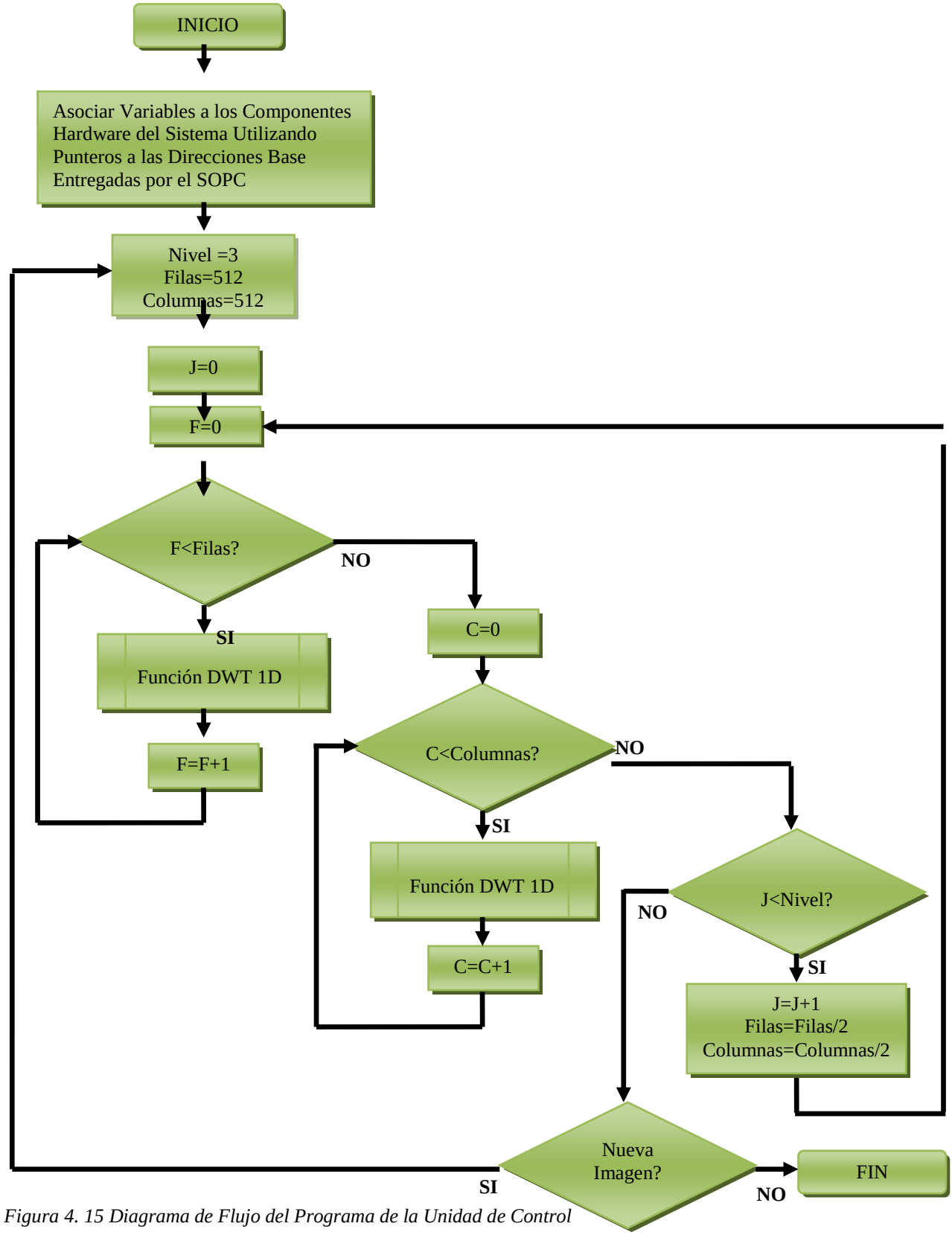


Figura 4. 15 Diagrama de Flujo del Programa de la Unidad de Control

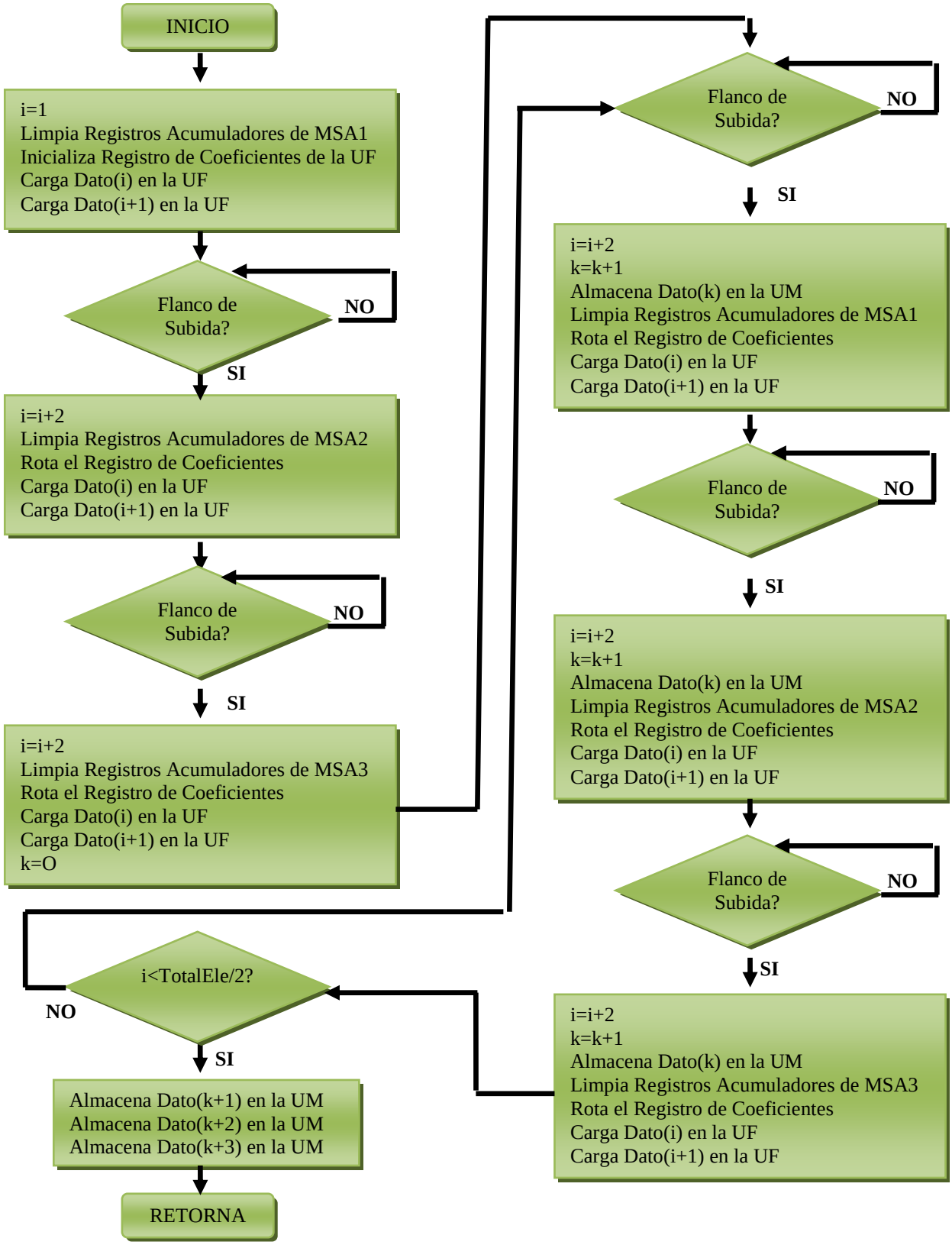


Figura 4. 16 Diagrama de Flujo de la Función para el Cálculo de la Transformada en una Dimensión

Capítulo 5.

Codificadores basados en la Transformada Wavelet

En este capítulo se presenta la etapa de codificación del sistema, el capítulo inicia con una revisión de algunas de las técnicas de codificación de imágenes más representativas, seguido por un análisis comparativo y finalmente se presenta una propuesta de modificación del algoritmo SPIHT.

5.1.- TÉCNICAS DE CODIFICACIÓN

En la última etapa del sistema de compresión la información que ya ha sido transformada para concentrar la energía de la señal en un conjunto de píxeles más cerrado, es ahora procesada para reducir el número de bits necesarios para representar la imagen. Entre los codificadores basados en transformada wavelet más empleados se pueden mencionar: EZW (Embedded Zerotree Wavelet), SPIHT (Set Partitioning in Hierarchical Trees), EBCOT (Embedded Block Coding whit Optimal Truncation Points) y LTW (Wavelet Lower Trees). Aunque existen también numerosas adaptaciones de estos algoritmos [42] [43] [44] [45] centraremos nuestra atención en los primeros por ser los más representativos.

5.1.1- Codificación EZW

El algoritmo EZW fue propuesto por Shapiro [38], inicialmente se encuentra el mayor valor absoluto de la matriz de coeficientes de la imagen y se calcula el número de bits necesarios para su representación, el umbral de partida se establece como el mayor valor que se puede representar con ese número de bits menos uno. Después se recorre la imagen en zig-zag y se construye un mapa de significancia, un coeficiente es significante si es mayor al umbral, esta información se almacena en unas estructuras denominadas árboles de ceros (zerotree). Los zerotree organizan los coeficientes de cuatro en cuatro, cada coeficiente tiene cuatro hijos y a su vez cada hijo tiene cuatro hijos. El recorrido inicia en los coeficientes de mayor nivel de transformación, el algoritmo aprovecha esta estructura puesto que por lo general los coeficientes se decrementan a medida que aumenta la escala y los hijos tienden a tener magnitudes menores a los padres. La significancia se etiqueta de la siguiente manera: ZTR si el valor absoluto del coeficiente y el valor absoluto de todas los coeficientes de su árbol es menor al umbral; IZ si el valor absoluto del coeficiente es menor al umbral y existe al menos un coeficiente de su árbol que es mayor al umbral; POS, si es mayor al umbral y positivo; NEG, si es mayor al umbral y negativo. Una vez se ha recorrido toda la imagen se decrementa el umbral y se repite el proceso hasta alcanzar el umbral cero, en cada decremento del umbral se codifica un plano de bits.

5.1.2- Codificación SPIHT

SPITH fue desarrollado por Said y Pearlman [39], quienes modificaron el algoritmo EZW presentando una estructura mejorada del árbol donde se tiene en cuenta la jerarquía y se utilizan conjuntos de datos organizados en subárboles. En la construcción de los árboles se mantiene la relación de herencia entre los coeficientes wavelet y se transmite la significancia entre la descendencia. La información de la significancia se almacena en tres listas ordenadas:

LIP, lista de píxeles no significativos, contiene las coordenadas de los coeficientes no significativos con respecto al umbral.

LSP, lista de píxeles significativos, contiene las coordenadas de los coeficientes significativos con respecto al umbral.

LIS, lista de conjuntos no significativos, contiene las coordenadas de las raíces de los árboles y subárboles no significativos con respecto al umbral. Cada raíz tiene además una marca que indica si representa a todos los descendientes o solo a los hijos de los hijos.

El algoritmo se desarrolla en cuatro pasos: el paso de inicialización donde se define el umbral de partida a partir del mayor valor absoluto de los coeficientes de la imagen y se inicializan las listas, el paso de clasificación donde se ordenan los coeficientes en las listas de acuerdo a la significancia, el paso de refinamiento donde se excluye la información que ya ha sido codificada y el paso de cuantificación donde se decrementa el umbral de uno en uno, en cada decremento del umbral se codifica un plano de bits, si el umbral es diferente de cero se salta al paso de clasificación, el proceso se repite hasta alcanzar el umbral cero.

El algoritmo por si solo disminuye la cantidad de bits necesarios para representar la imagen, sin embargo, la tasa de compresión puede mejorarse aplicando sobre el flujo binario de salida un codificador aritmético o cualquier otro que sea orientado al manejo de grandes cantidades de datos binarios.

5.1.3- Codificación LTW

El algoritmo LTW fue presentado por Oliver y Malumbres [40], éste conserva la estructura de árbol presentada en SPIHT. EL algoritmo se desarrolla en dos pasos:

Paso 1: Los coeficientes son etiquetados de acuerdo a la significancia. Para determinar si un coeficiente es significante el algoritmo utiliza un umbral que corresponde a un proceso de cuantificación que elimina los planos de bits que corresponden a la parte menos significativa de los coeficientes, un coeficiente es insignificante si está por debajo del umbral. Se recorre la matriz de hijos a padres en grupos de cuatro y se etiqueta con el símbolo L si un coeficiente y su descendencia son insignificantes, se marca con el símbolo I si un coeficiente es insignificante pero su descendencia no los es; cuando el coeficiente y

alguno de sus hijos son significantes se utiliza un símbolo que indique el número de bits necesarios para representar el valor del coeficiente y el signo, cuando el coeficiente es significante pero sus cuatro hijos no lo son se etiqueta el coeficiente con un símbolo que contiene información que indica el número de bits necesarios para representar el valor del coeficiente seguido del signo y el indicador L.

Paso 2. Los valores de los coeficientes son codificados usando los símbolos obtenidos en el primer paso y un codificador aritmético.

5.1.4- Codificación EBCOT

EBCOT fue propuesto por Taubman [41], en éste cada subbanda es dividida en bloques llamados code-blocks los cuales son codificados por planos de bits, para cada plano de bits se desarrollan tres pasos de codificación: refinamiento de la magnitud, propagación de la significancia y normalización, cada bit del plano es codificado en solo uno de los tres pasos. En el paso de refinamiento de la magnitud se codifican los bits que pertenecen a un coeficiente que ha sido codificado como significante en planos previos, el paso de propagación de la significancia es usado para expandir la información de significancia y en el paso de normalización se codifican los bits no significativos que no tienen en su vecindad bits significativos. El algoritmo se fundamenta en el uso de cuatro operaciones primitivas que forman la base de la estrategia de codificación estas son ZC (zero coding), RLC (run-length coding), SC (sign coding) y MR (magnitudes refinement). Si una muestra no ha sido codificada aun como significante, se utilizan las primitivas ZC y RLC para codificar si es o no es significante, si lo es la primitiva SC es también invocada para identificar el signo; si la muestra ya es significante la primitiva MR es usada para codificar el valor. Los datos de codificación obtenidos del proceso anterior se codifican usando un codificador aritmético binario, (el codificador binario MQ). Finalmente el flujo de bits se organiza en paquetes, donde los menos significativos se pueden descartar para mejorar la tasa de bits aumentando la distorsión. Este algoritmo es usado en el estándar de compresión de imágenes JPEG200.

5.2.- ALGORITMO DE CODIFICACIÓN BASADO EN SPIHT

Como se mencionó anteriormente, aunque existen numerosas publicaciones de algoritmos de codificación fundamentados en el uso de la transformada wavelet para la implementación presentada en este trabajo seleccionamos entre cuatro de los algoritmos más representativos; se presentó EZW debido a que constituye el inicio de la generación de codificadores que utilizan estructuras de árboles de ceros, se consideró el codificador SPIHT que al ser un algoritmo de complejidad muy baja y tasas de compresión satisfactorias es el mas utilizado, el algoritmo EBCOT es un referente obligatorio puesto que fue seleccionado como codificador central del estándar JPEG2000, también se presenta el codificador LTW por ser uno de los desarrollos mas recientes.

La figura 5.1 grafica los resultados de la comparación de desempeño presentados en [40], la distorsión es medida por el pico de la relación señal a ruido (PSNR) y se grafica versus la tasa de compresión entendida como la relación entre el número de bits que se requieren para representar la imagen de forma comprimida y el número de bits de la imagen original, los algoritmos fueron aplicados sobre la clásica imagen de prueba "*Lena*", una imagen de textura "*VisTex*" y una imagen utilizada para el test de JPEG2000 "*Woman*". De la figura 5.1 se puede observar que: para las imágenes analizadas SPIHT y EBCOT mantienen niveles de PSNR similares, en los tres casos LTW presenta la mejor relación de desempeño, EZW solo fue evaluado en la imagen "*Lena*" y su desempeño fue mejorado por los otros codificadores. Sin embargo, es importante resaltar que EZW es un algoritmo que se caracteriza por su baja complejidad; SPIHT mantiene la simplicidad de EZW pero maneja una estructura del árbol que le permite alcanzar un mejor desempeño; EBCOT es un compresor muy versátil que permite indicar el tipo de escalabilidad deseada y el tamaño exacto del fichero final, pero esta particulares aumentan su complejidad y costo computacional; el primer paso de LTZ es de baja complejidad pero en el segundo paso requiere que los datos entregados sean codificados mediante un codificador aritmético lo que traslada el problema a la búsqueda de un modelo eficiente para la implementación de dicho codificador. Para la implementación del sistema presentado en este trabajo se

seleccionó el algoritmo de codificación SPIHT puesto que logra un balance entre complejidad y desempeño.

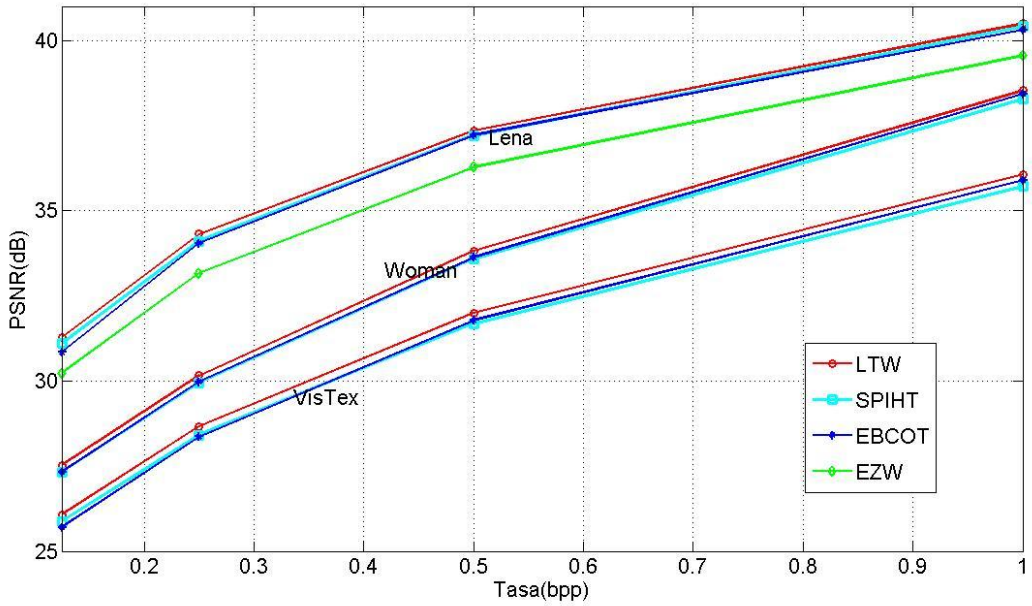


Figura 5. 1 PSNR de los codificadores LTW, SPIHT, EBCOT y EZW a diferentes tasas de compresión [9]

5.2.1- Estructura del Árbol

En SPIHT los datos se organizan en estructuras de árbol de orientación espacial. La figura 5.2 muestra su relación para tres niveles de transformación wavelet; cada coeficiente de F_{LH}, F_{HL} y F_{HH} del nivel superior tiene cuatro hijos del nivel inmediatamente inferior, y a su vez cada hijo tiene cuatro hijos en el siguiente nivel, los coeficientes de F_{LL} del nivel superior no tienen descendencia, esta relación se muestra en la gráfica a través de colores.

[9] Imagen construida con datos tomados de:
J. Oliver, M.P. Malumbres, "Low complexity multiresolution image compression using wavelet lower trees," IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 11, pp. 1437 – 1444, noviembre de 2006.

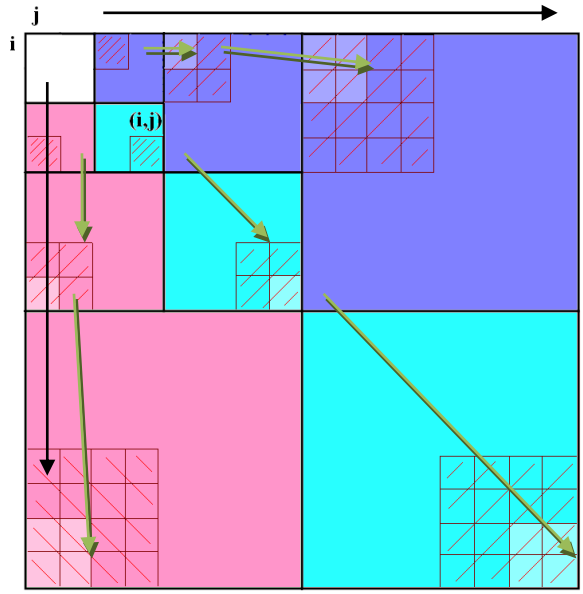


Figura 5. 2 Estructura de Árbol Jerárquico

La notación utilizada [39] para describir la distribución del árbol, siempre que las coordenadas estén dentro de los límites de la imagen es:

$$O(i,j)=\begin{Bmatrix}(2i,2j) & (2i,2j+1)\\ (2i+1,2j) & (2i+1,2j+1)\end{Bmatrix} \qquad \text{EC.5.1}$$

$$L(i,j)=\begin{Bmatrix}(4i,4j) & (4i,4j+1) & (4i,4j+2) & (4i,4j+3)\\ (4i+1,4j) & (4i+1,4j+1) & (4i+1,4j+2) & (4i+1,4j+3)\\ (4i+2,4j) & (4i+2,4j+1) & (4i+2,4j+2) & (4i+2,4j+3)\\ (4i+3,4j) & (4i+3,4j+1) & (4i+3,4j+2) & (4i+3,4j+3)\end{Bmatrix} \qquad \text{EC.5.2}$$

$$D(i,j)=L(i,j)+O(i,j) \qquad \text{EC.5.3}$$

Donde (i, j) son las coordenadas de la raíz del árbol, el conjunto O(i, j) representa los hijos del nodo y L(i, j) representa los hijos de los hijos. El conjunto D(i, j) representa toda la descendencia del nodo.

La significancia S de un conjunto se calcula a partir de la siguiente definición:

$$S(\tau)=\begin{cases}1 & \text{Cuando al menos un elementos del conjunto } \tau \text{ es mayor o igual al umbral}\\ 0 & \text{en otro caso}\end{cases}$$

5.2.2- Algoritmo SPIHT

Los pasos del seudocódigo del algoritmo de codificación SPIHT presentado en [39] se presentan a continuación:

1) **Inicialización**:
-Se define el umbral como 2^{n-1} donde n es el mínimo número de bits necesarios para representar el mayor valor absoluto de la matriz de coeficientes
-Colocar la lista LSP como una lista vacía
-Ingresar a la lista LIP las coordenadas de los coeficientes del mayor nivel de transformación
-Ingresar a la lista LIS las coordenadas de los coeficientes de F_{LH}, F_{HL} y F_{HH} del nivel de transformación superior y marcarlas como Tipo A

2) **Paso de Clasificación**:
2.1) Para cada entrada (i, j) en la lista LIP hacer:
2.1.1) Sacar S((i, j));
2.1.2) Si S(i, j) = 1 entonces mover (i, j) a la lista LSP y sacar el signo
2.2) Para cada entrada (i, j) en la lista LIS hacer:
2.2.1) Si la entrada es de tipo A entonces
Sacar S(D(i, j));
Si S(D(i, j)) = 1 entonces
Para cada (k, l) $\in$O(i, j) hacer:
Sacar S(k, l);
Si S(k, l) = 1 entonces Agregar (k, l) a LSP y sacar el signo
Si S(k, l) = 0 entoncesAgregar (k, l) a LIP
Si L(i, j) # 0 entonces
mover(i, j) al final de la lista LIS, como una entrada de tipo B
Saltar al paso 2.2.2
de otra forma
Eliminar (i, j) de la lista LIS
2.2.2) Si la entrada es de tipo B entonces
Sacar S(L(i, j));
Si S(L(i, j)) = 1 entonces
Agregar cada (k, l) $\in$ O(i, j) al final de la lista LIS como entrada tipo A
Eliminar (i, j) de la lista LIS.

3) **Paso de Refinamiento**: Para cada entrada (i, j) en la lista LSP, eliminar de los coeficientes los bits que ya han sido codificados
4) **Paso de Cuantificación**: Decrementar n en 1 y saltar al paso 2.

5.2.3- Modificaciones Propuesta para el Algoritmo SPIHT

En el algoritmo anterior para el flujo de salida se utilizan cuatro símbolos cero (0) o uno (1)

para indicar la significancia, positivo (+) o negativo (-) para indicar el signo del coeficiente, para la representación de estos cuatro símbolos si no se utiliza alguna técnica de codificación adicional se requieren 2 bits por símbolo; la propuesta de mejora del algoritmo consiste en codificar al inicio los signos de cada coeficiente, de forma tal que el flujo de salida solo requiera de dos símbolos, para lo cual solo se requiere de un bit por símbolo mejorando la tasa de compresión.

En el paso de inicialización se agregan a la lista LIP todos los coeficientes del nivel mayor de transformación y posteriormente en el paso de clasificación si son significantes se mueven de LIP a LSP, lo que requiere un reordenamiento de la lista LIP; de otro lado la información de la lista LIP no se utiliza en pasos posteriores, por lo tanto se elimina la lista LIP y se plantea entonces realizar en el paso de clasificación una validación previa para asignar a la lista LSP solo los coeficientes de nivel superior que corresponden.

Los pasos del seudocódigo del algoritmo de codificación SPIHT modificado se presentan a continuación:

1) **Paso de Codificación del signo**:
1.1) Para cada entrada (i, j) en la matriz de coeficientes hacer
    Si el coeficiente (i, j) es positivo entonces NS(i, j)=0 y Enviar a la Salida
    Si el coeficiente (i, j) es negativo entonces NS(i, j)=1 y Enviar a la Salida
2) **Inicialización**:
    -Se define el umbral como 2^{n-1} donde n es el mínimo número de bits necesarios para representar el mayor valor absoluto de la matriz de coeficientes
    -Colocar las listas LSP como lista vacía
    -Ingresar a la lista LIS las coordenadas de los coeficientes de F_{LH}, F_{HL} y F_{HH} del nivel de transformación superior y marcarlas como Tipo A
3) **Paso de Clasificación**:
3.1) Para cada coeficiente del mayor nivel de transformación con coordenadas (i, j) hacer:
    3.1.1) Calcular y Enviar a la Salida S(i, j);
    3.1.2) Si S(i, j) = 1 entonces agregar (i, j) a la lista LSP
3.2) Para cada entrada (i, j) en la lista LIS hacer:
    3.2.1) Si la entrada es de tipo A entonces
        Sacar S(D(i, j));
        Si S(D(i, j)) = 1 entonces
            Para cada (k, l) $\in$O(i, j) hacer:
                Sacar S(k, l);
                Si S(k, l) = 1 entonces Agregar (k, l) a LSP
            Si L(i, j) # 0 entonces
                mover(i, j) al final de la lista LIS, como una entrada de tipo B

Saltar al paso 3.2.2
de otra forma
Eliminar (i, j) de la lista LIS
3.2.2) Si la entrada es de tipo B entonces
Sacar S(L(i, j));
Si S(L(i, j)) = 1 entonces
Agregar cada (k, l) ∈ O(i, j) al final de la lista LIS como entrada tipo A
Eliminar (i, j) de la lista LIS.
4) **Paso de Refinamiento**: Para cada entrada (i, j) en la lista LSP, eliminar de los coeficientes los bits que ya han sido codificados
5) **Paso de Cuantificación**: Decrementar n en 1 y saltar al paso 3.

Para su evaluación los algoritmos *SPIHT* y *SPIHT modificado* se codificaron en Matlab (el código fuente puede ser consultado Anexo A), se seleccionaron como imágenes de prueba "*lena*", "*gantrycrane*", "*pears*", "*peppers*" y "*liftingbody*", disponibles en el toolbox de imágenes de Matlab, la tabla 5.1 muestra la tasa de compresión obtenida para cada imagen de prueba ajustada a un tamaño de 256x256 píxeles con 16 bpp en escala de gris, estas imágenes fueron transformadas a tres niveles con la base wavelet biortogonal (5,3). Para evaluar únicamente el desempeño de los algoritmos se calculó la tasa de compresión como el cociente entre el total de bits necesarios para representar la imagen comprimida y el total de bits necesarios para representar la imagen transformada.

Tabla 5. 1 Tasa de Compresión SPIHT y SPIHT modificado

IMAGEN DE PRUEBA	**TASA DE COMPRESIÓN (bpp)**	
	SPIHT	**SPIHT MODIFICADO**
lena	0.90	0.55
gantrycrane	0.78	0.49
pears	0.86	0.53
peppers	0.73	0.46
liftingbody	0.76	0.48

La modificación del algoritmo se establece básicamente en la forma en que se representa la

información del signo por lo tanto la calidad de la imagen es igual en los dos algoritmos y como se puede observar en los resultados registrados en la tabla 5.1 con esta modificación se logra una mejor tasa de compresión, cabe destacar que en ambos algoritmos es posible utilizar además un codificador orientado al manejo de grandes cantidades de datos binarios para mejorar la tasa de compresión.

Para la implementación en hardware el algoritmo SPIHT modificado se desarrolla en C++ y se ejecuta al interior de una FPGA utilizando un diseño soportado en el sistema Nios II, el código fuente puede ser consultado en el anexo C.

5.2.4- Decodificador SPIHT modificado

El decodificador realiza el proceso inverso del codificador, en el algoritmo modificado los dos primeros bytes que llegan corresponden al tamaño de la imagen N y al nivel de transformación, los siguientes NxN bits contienen la información del signo de los coeficientes, los restantes bits guardan la información de los planos de bits iniciando por el plano mas significativo de esta forma los siguientes $[N/(2\text{^}(Nivel-1))]^2$ bits corresponden a los coeficientes de nivel superior del plano mas significativo y para los siguientes bits se evalúa si el bit es cero entonces corresponde a un árbol de ceros y toda la descendencia de esta raíz se coloca en cero, si el bit es uno entonces los siguientes cuatro bits corresponden a los hijos de esa raíz, el proceso se repite hasta completar los NxN bits del plano, entonces se replica el proceso para el siguiente plano hasta llegar al plano menos significativo. En SPIHT el orden de llegado de los bits es fundamental para la reconstrucción de la matriz.

Al igual que el codificador, para la implementación en hardware el decodificador SPIHT se desarrolla en C++ y se ejecuta al interior de una FPGA utilizando un diseño soportado en el sistema Nios II, el código fuente puede ser consultado en el anexo C.

Capítulo 6.

Pruebas y Resultados

En este capítulo se presenta una síntesis de los resultados de las principales pruebas realizadas sobre el sistema de compresión de imágenes con el fin de verificar la funcionalidad y el comportamiento del compresor. Se realizaron pruebas de desempeño para diferentes niveles de transformación, para varios umbrales de cuantificación y al final del capitulo se presentan los resultados de las pruebas realizada sobre una propuesta para codificación de video.

6.1.- MEDIDAS DE CALIDAD

Para la evaluación del sistema es necesario presentar medidas cuantitativas que determinen la calidad de los datos obtenidos, en la evaluación de sistemas de compresión el pico de la relación señal a ruido (PSNR) es ampliamente usada para medir la distorsión introducida.

El pico de la relación señal a ruido esta definido por:

$$PSNR(x, y) = 10 \log_{10}\left(\frac{x_{\max}^2}{MSE}\right) \text{dB} \qquad \text{EC 6.1}$$

Donde x_{max} es el máximo valor que puede tomar un píxel y depende del número de bits, MSE es el error cuadrado medio y esta definido por:

$$MSE(x,y) = \frac{1}{N}\sum_{x,y}\left[f(x,y) - f'(x,y)\right]^2 \qquad \text{EC 6.2}$$

Donde *N* es el numero de píxeles de la imagen, *f* es la imagen original y *f'*es la imagen reconstruida.

Para caracterizar la capacidad de compresión del sistema también se usa la tasa de compresión, entendida como la relación entre el número de bits que se requieren para representar la imagen de forma comprimida y el número de bits de la imagen original, esta tasa se mide en unidades de bit por píxel (bpp).

6.2.- CALIDAD DE LA SEÑAL PARA DIFERENTES NIVELES DE TRANSFORMACION

Las figuras 6.1 y 6.2 muestran el desempeño del sistema usando diferentes niveles de transformación wavelet aplicado en las imágenes de prueba "*Lena*", "G*antrycrane*", "*Pears*", "*Peppers*" y "*Liftingbody*" disponibles en el toolbox de imágenes de Matlab; en la figura 6.1 se grafica la tasa de compresión versus el nivel de transformación, se puede observar que entre mas niveles de transformación se utilicen se requieren menos bits para representar la información de la imagen, sin embargo en la figura 6.2 donde se grafica el pico de la relación señal a ruido (PSNR) versus el nivel de transformación se observa que entre mas niveles de transformación se utilicen se aumenta la distorsión en la señal de salida, esto se debe a la aproximación que se presenta al usar solo números enteros en las diferentes etapas del sistema de compresión. Teniendo en cuenta este comportamiento podemos afirmar que entre mas niveles de transformación se apliquen mejora la tasa de compresión pero se disminuye la calidad de la imagen. En la grafica 6.1 también se observa que usando solo un nivel de transformación no se presenta compresión, por el contrario se requieren más bits para representar la información, esto se debe a que en el primer nivel de transformación no es posible asociar árboles, de acuerdo al algoritmo todos los píxeles se consideran raíz y deben ser codificados. Según lo planteado en [52] cuando

la PSNR es de 40 dB o mas, la imagen original y la reconstruida son indistinguibles por observadores humanos, en el diseño presentado para la compresión se recomienda el uso de 2 o 3 puesto que en estos niveles la relación señal a ruido esta por encima de los 40dB.

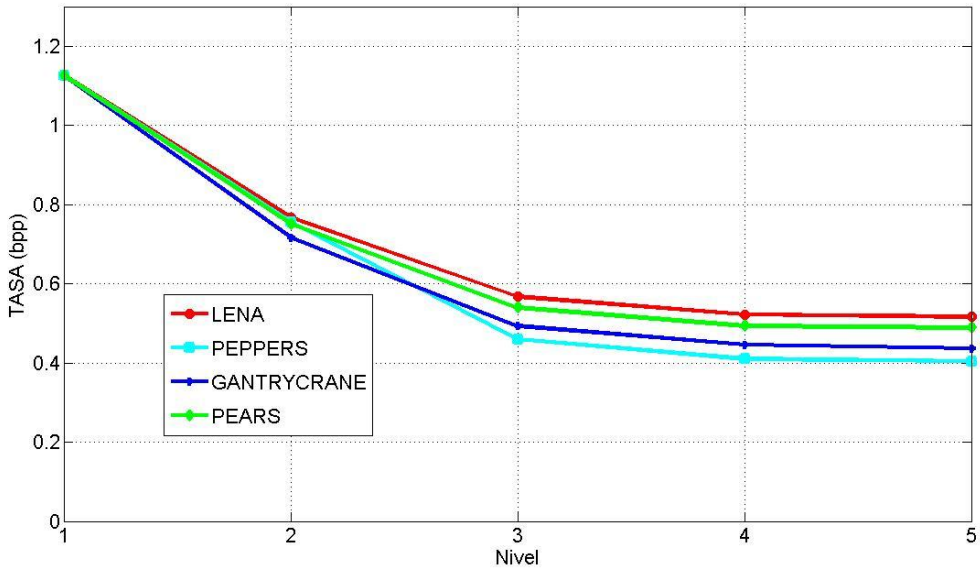


Figura 6. 1 Tasa de Compresión Para Diferentes Niveles de transformación

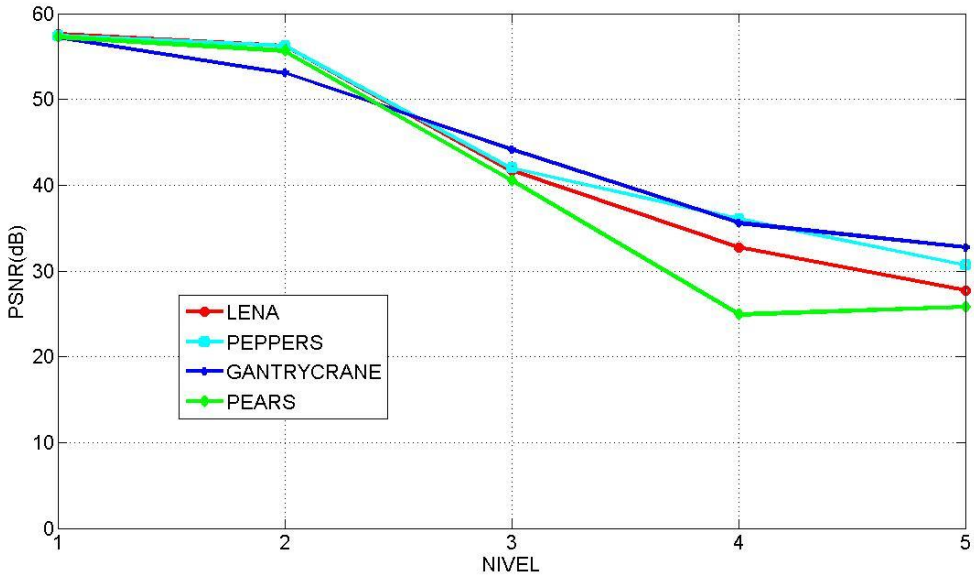


Figura 6. 2 Relación Señal a Ruido Para Diferentes Niveles de Transformación

6.2.- CALIDAD DE LA SEÑAL PARA DIFERENTES UMBRALES DE CUANTIFICACION

La figura 6.3 muestra el desempeño del sistema usando diferentes umbrales de cuantificación y dos niveles de transformación para la compresión de las imágenes de prueba "*Lena*", "G*antrycrane*", "*Pears*", "*Peppers*" y "*Liftingbody*", donde Q=1 indica que la compresión se realizó sin tener en cuenta la información almacenada en el bit menos significativo de los coeficientes wavelet, Q=2 indica que la compresión se realizó sin tener en cuenta la información de los dos bits menos significativos, de igual forma se realizaron pruebas para Q=3 y Q=4. En la gráfica se observa que cuando se descartan más bits de información la tasa disminuye lo que indica que se requieren menos bits para representar la información pero implica que la relación señal a ruido disminuya lo que indica que existe una desmejora en la calidad de la imagen. En el diseño del compresor presentado se recomienda usar cuantificación con Q menor o igual a 2 debido a que en estos umbrales la relación señal a ruido esta por encima de los 40 dB.

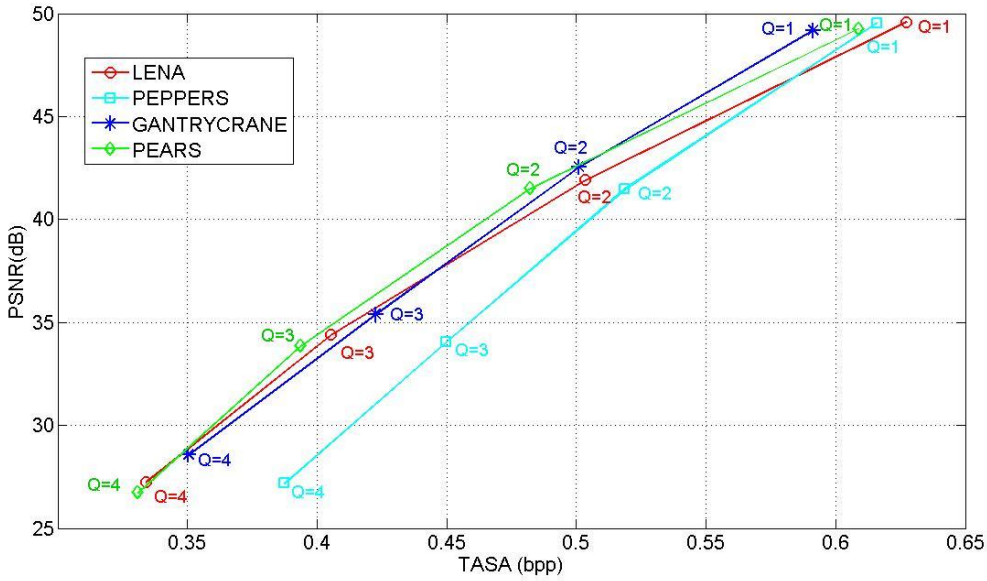


Figura 6. 3 Relación Señal a Ruido y Tasa de compresión Para DiferentesUmbrales de cuantificación

6.3.- MANEJO DE VÍDEO

La compresión de video es el proceso de compactar una secuencia de imágenes en un número reducido de bits. Un modelo simple de compresor de video es utilizado por Motion JPEG (M-JPEG) [53] donde cada imagen de la secuencia es comprimida de manera individual, en este modelo todas las imágenes de la secuencia tendrán garantizadas la misma calidad y requieren el mismo tiempo de procesamiento para su compresión. Existen métodos más elaborados que toman provecho de la relación que existe entre las imágenes de la secuencia como el presentado en grupo de estándares MPEG [54] donde se usa un modelo basado en la predicción con compensación de movimiento, el principio básico es codificar una imagen de referencia (trama I) y para las siguientes imágenes codificar tan solo las partes que difieren de una trama anterior (trama P) o las diferencias entre una trama anterior o posterior (trama B), esto aumenta la complejidad al tiempo que mejora la tasa de compresión. Existen numerosas publicaciones que proponen esquemas alternativos de compresión de video en el dominio wavelet; en [55] proponen un esquema de compresión donde la secuencia de vídeo es primero transformada usando una estructura wavelet 3D con o sin compensación de movimiento y entonces es codificada usando una extensión en tres dimensiones del algoritmo SPIHT en [62] proponen un esquema similar; en [56] presentan una estructura en la que se realiza estimación y compensación de movimiento sobre los detalles de la imagen transformada; en [57] plantean el uso de una técnica de compresión donde cada imagen es descompuesta usando transformación wavelet, posteriormente una operación de detección de movimiento divide los árboles en árboles con movimiento y árboles sin movimiento, los árboles sin movimiento son codificados con una decisión binaria y los árboles con movimiento son codificados usando un esquema de codificación fractal; en [58] presentan una técnica llamada Codificación Wavelet Flexible en Bloques (FBWC) donde primero una compensación de movimiento elimina la redundancia temporal entre tramas adyacentes y se generan tramas de error predictivo, después estas tramas de error son transformadas en el dominio wavelet usando bloques flexibles y finalmente un mecanismo de codificación de árboles de ceros y un codificador de entropía son usados para comprimir los coeficientes wavelet; en [59]

exponen una técnica de compresión de imágenes diferencial, el concepto básico es desarrollar transformada wavelet sobre la primera trama, que después es cuantificada, codificada y almacenada, todas las tramas siguientes son transformadas de la misma forma pero son comparadas con los coeficientes almacenados previamente y solo serán transmitidas si superan un umbral de referencia; otros autores [60] [61] [63] presentan técnicas similares que utilizan una ampliación de la transformada wavelet en tres dimensiones en sus esquemas de compresión. Sin embargo en este trabajo la complejidad de las técnicas mencionadas anteriormente nos lleva a la elección del modelo presentado por Hilton [16] que utiliza un codificador de video simple mostrado en la figura 6.4.

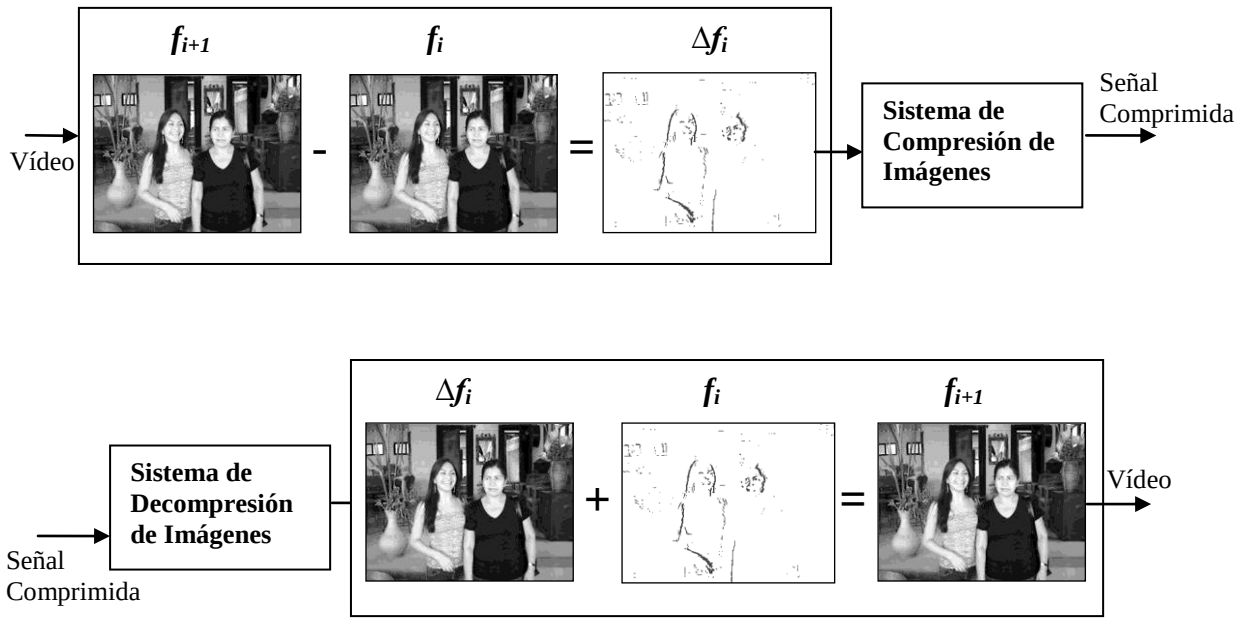


Figura 6. 4 Esquema del Sistema de Compresión de Vídeo

Donde la señal de video esta expresada como una secuencia de imágenes f_i para i=0,1,…, donde f_i representa la i-esima imagen de la secuencia y la diferencia entre dos imágenes adyacentes esta dada por:

$$\Delta f_i = f_{i+1} - f_i \qquad \text{EC 6.3}$$

Δf_i es llamada imagen de diferencias y contiene solo los cambios entre dos imágenes consecutivas, con este método las regiones que son comunes a dos imágenes sucesivas solo se procesaran una vez lo que conlleva a una mejora en la tasa de compresión. Se realizaron pruebas de este sistema de compresión de video para las tres secuencias (figuras 6.5, 6.6 y 6.7) con una muestra de 16 imágenes de la secuencia, usando dos niveles de transformación y un umbral inferior de cuantificación con Q=1, los resultados se muestran en la tabla 6.1.

Tabla 6. 1 Tasa de Compresión y Relación Señal a Ruido en el Sistema de Compresión de Vídeo

VÍDEO	**TASA**	**PSNR**
SECUENCIA 1	0,6277	26,0362
SECUENCIA 2	0.6310	28,364
SECUENCIA 3	0.6211	36,1376

Estos resultados se pueden mejorar incorporando técnicas mas robustas para la eliminación de la redundancia temporal, sin embargo este desarrollo se plantea como una continuación de este trabajo de investigación.

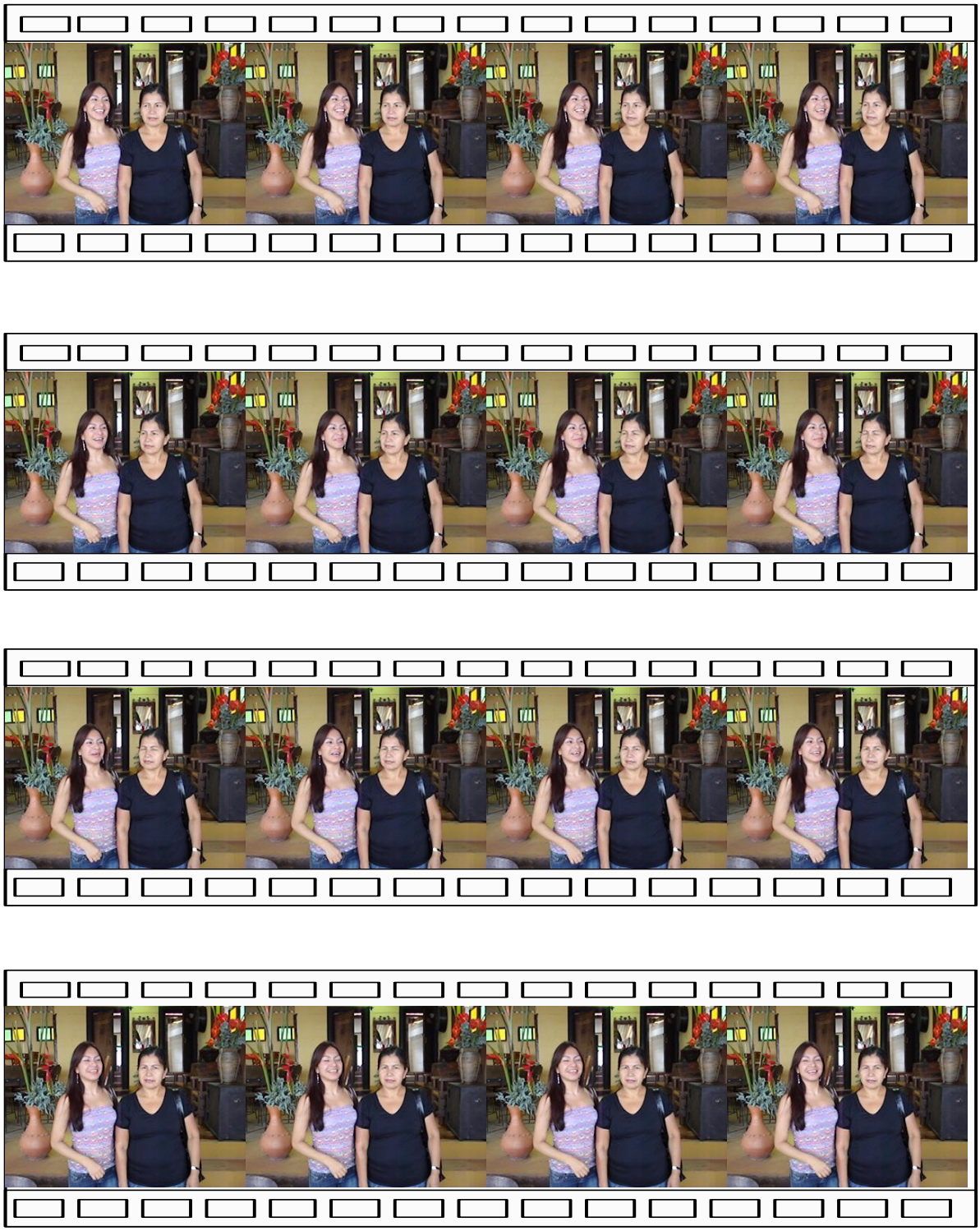


Figura 6. 5 Secuencia 1 Utilizada Para las Pruebas del Sistema de Compresión de Vídeo

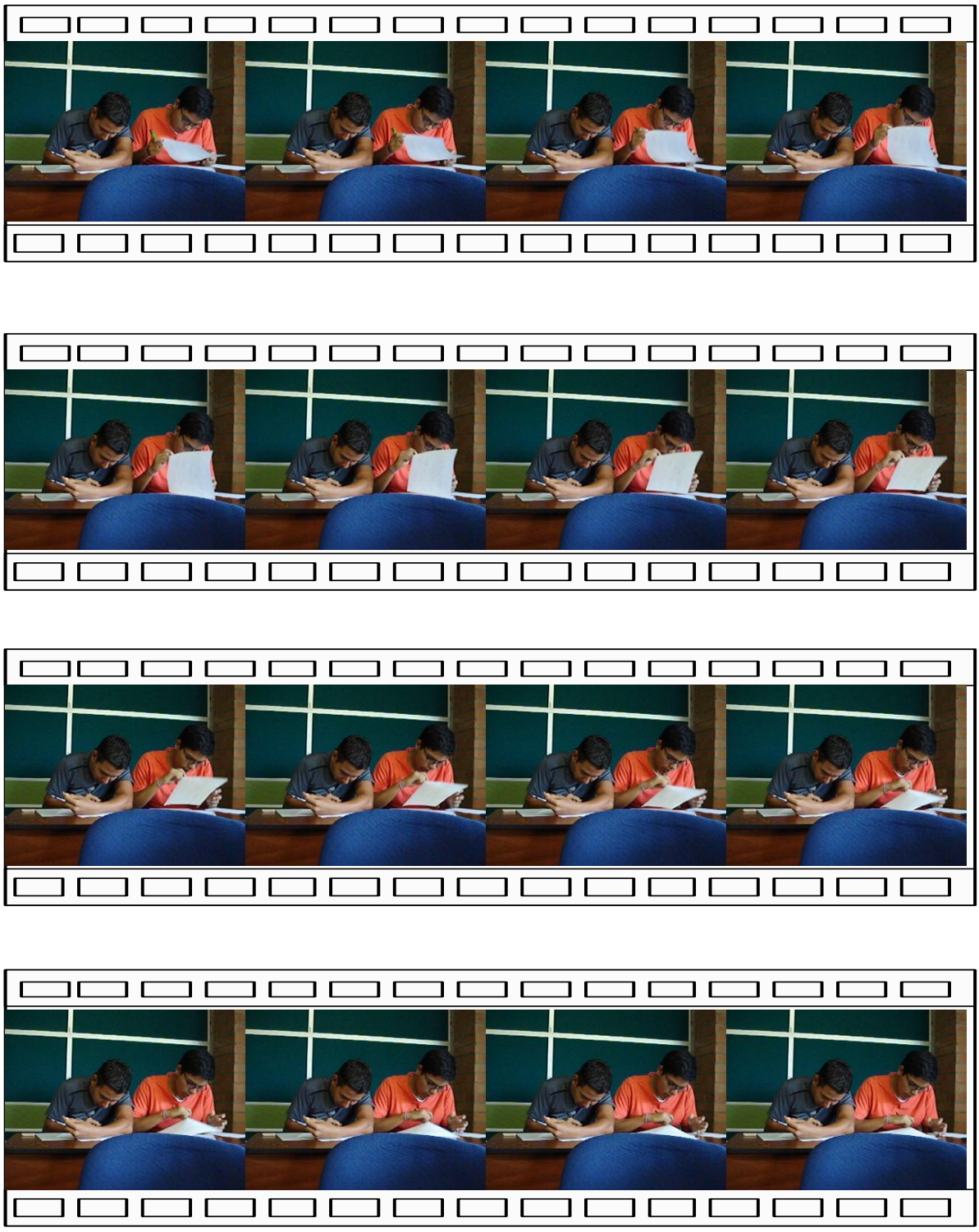


Figura 6. 6 Secuencia 2 Utilizada Para las Pruebas del Sistema de Compresión de Vídeo

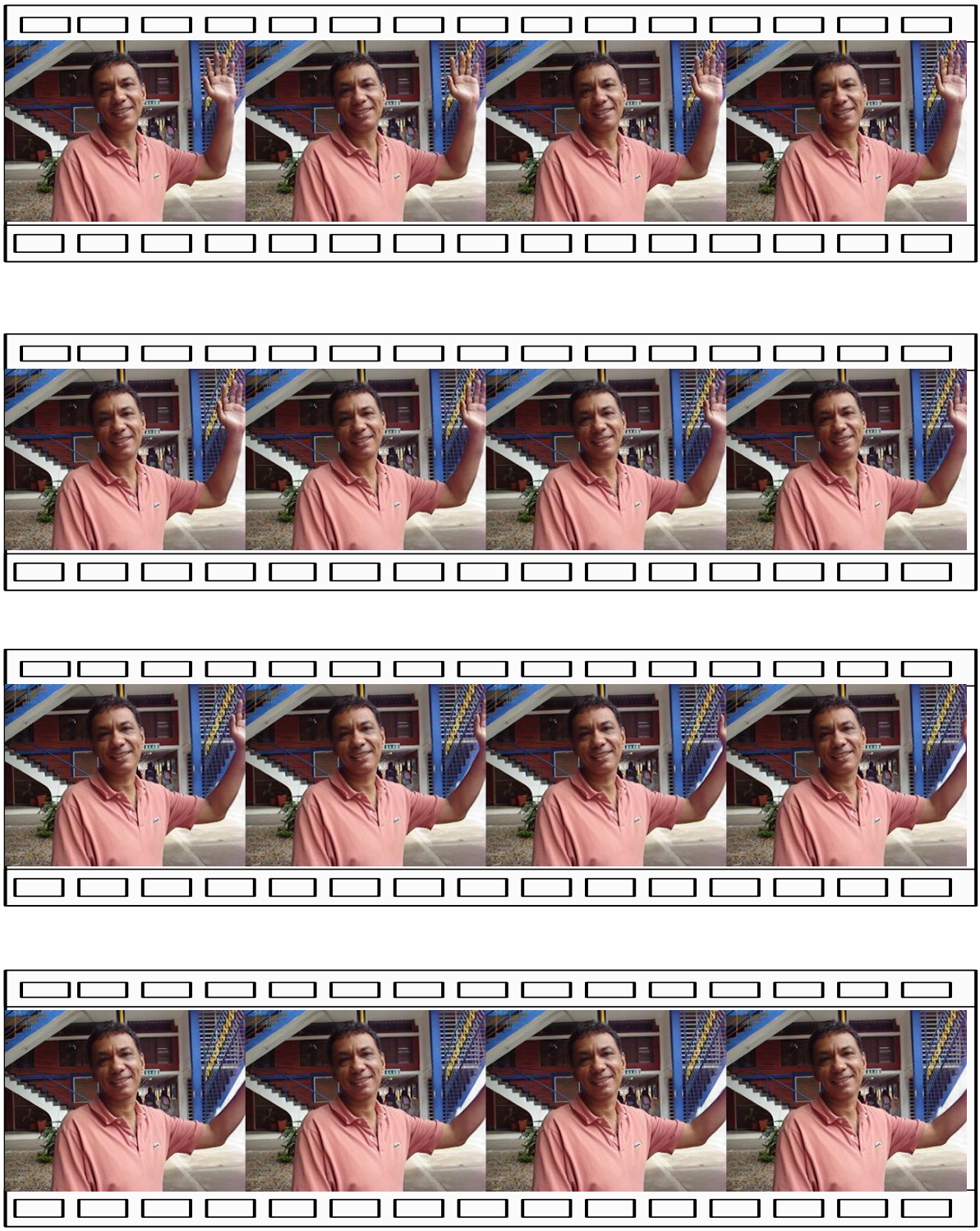


Figura 6. 7 Secuencia 3 Utilizada Para las Pruebas del Sistema de Compresión de Vídeo

Capítulo 7.

Conclusiones y Líneas de Investigación Futuras

7.1.- CONCLUSIONES.

Se ha manifestado la necesidad de compactar la información de imágenes en forma digital debido a su elevado volumen. El sistema propuesto para la compresión emplea el método de codificación de transformadas y maneja los componentes de color de forma separada, en este modelo los pixeles de la imagen son transformados al dominio wavelet para posteriormente ser codificadas y reducir el número de bits necesarios para su representación.

Para obtener un mejor desempeño se ajustan los datos de entrada de forma que halla una distribución en todo el rango de los posibles valores de entrada, de igual forma se realiza una transformación del modelo de color para representar la imagen en el modelo YUV que responde mejor al sistema de visión humano.

En aplicaciones de compresión la transformada wavelet se considera más apropiada que la transformada coseno puesto que sus características se asemejan más a la respuesta del sistema de visión humano por lo que se obtienen mejores tasas de compresión; de otro lado debido a que la DWT no necesita partir la imagen de entrada en bloques no tiene el problema de correlación en los límites de los bloques que presenta la DCT, además la DWT es más robusta para la transmisión y decodificación de errores.

Para la implementación del codificador se seleccionó la base wavelet Bior5.3 debido a que posee características que la ubican entre las más recomendadas para aplicaciones de codificación, esta base posee un número reducido de coeficientes por lo que requiere menos operaciones para implementar la transformación y por ende se utiliza menos área, sus coeficientes pueden ser transformados a números enteros mediante una simple normalización por lo tanto no se requiere utilizar en la implementación módulos con aritmética en punto flotante disminuyendo el área necesaria para la implementación.

La principal contribución de esta tesis es la presentación de una arquitectura hardware para desarrollar la transformada discreta wavelet en dos dimensiones, la arquitectura utiliza $\frac{3}{4}N^2 + 3N$ ciclos para transformar una imagen de tamaño NxN logrando una mejora frente a las arquitecturas presentadas en trabajos previos. La implementación se realizó en una FPGA Ciclone II de Altera utilizando un diseño que combina módulos en VHDL y módulos de captura esquemática controlados por un sistema microcontrolado programado en C++ utilizando la plataforma de diseño Quartus y el procesador NIOS.

En este trabajo también se propone una modificación del algoritmo de codificación SPHIT que mejora la tasa de compresión, la modificación se realiza en la forma de representar la información del signo por lo tanto no se afecta la calidad de la imagen, también se elimina una de las listas con lo cual se disminuyen los requerimientos de almacenamiento, con la modificación propuesta se logra mantener la sencillez del algoritmo y se mejorar el desempeño medido en términos de la compresión.

El esquema de compresión de vídeo desarrollado es una buena elección debido a su simplicidad, lo que asegura una implementación con un consumo económico de recursos hardware, sin embargo para aplicaciones donde se requieren tasas de compresión mejores se recomienda el uso de un esquema que utilice técnicas mas robustas para la eliminación de la redundancia temporal.

7.2.- LÍNEAS DE INVESTIGACIÓN FUTURAS

En este trabajo de maestría se presenta un sistema de compresión de imágenes basado en la transformada wavelet implementado sobre una FPGA que provee un desempeño eficiente en área y velocidad de cálculo; sin embargo, es posible mejorar el sistema a través de investigaciones posteriores que pueden estar enfocadas a:

Se realizó una revisión de las bases wavelet existentes y sus características para determinar la más apropiada para esta aplicación, una investigación posterior podría desarrollarse en la construcción de una base wavelet pensada para sistemas de compresión que logre representar en un conjunto más cerrado la información importante de la imagen permitiendo alcanzar mejores tasas de compresión.

Se realizó una revisión del estado del arte de las técnicas de compresión más utilizadas, entre las cuales se seleccionó la más apropiada para este desarrollo y se implementó en un procesador embebido en la FPGA, una investigación posterior podría enfocarse en el diseño de un algoritmo de compresión pensado para implementación hardware, que logre un balance entre área y desempeño, aprovechando el paralelismo propio de las arquitecturas hardware.

Se presenta una arquitectura hardware que desarrolla la transformada wavelet en dos dimensiones sobre una FPGA, esta arquitectura está adaptada a un sistema de compresión, sin embargo puede ser usada en cualquier aplicación que requiera análisis frecuencial en subbandas, una investigación posterior podría enfocarse en el desarrollo de un sistema modular que fácilmente pueda transportarse y que permita análisis frecuencial en subbandas de diferentes señales.

Se implementó un sistema completo de compresión de imágenes, sin embargo los datos son cargados desde un dispositivo de almacenamiento externo hacia la FPGA y de igual forma los resultados son almacenados en un dispositivo de almacenamiento, para la visualización de los resultados los datos son exportados y graficados utilizando otra herramienta, un

trabajo posterior debería llevar al desarrollo de una etapa de captura y visualización de imágenes que permita advertir de una manera sencilla los resultados obtenidos.

Se presentó un esquema de compresión de vídeo de implementación sencilla, una investigación posterior debería centrarse en mejorar el desempeño del sistema incorporando técnicas como predicción y compensación de movimiento o alguna más avanzada que permita eliminar de manera eficiente la redundancia temporal.

Bibliografía

[1] A. Otero, R. Tolosa, P. Venini, "Tecnologías de banda ancha MPEG".

[2] C. A. Ordoñez, "Formatos de Imagen Digital," Revista Digital Universitaria, Universidad Autónoma de México, vol. 5, no. 7, pp. 2-10, Mayo 2005, disponible online en http://www.revista.unam.mx/vol.6/num5/art50/may_art50.pdf consultado en diciembre 2010.

[3] A. N. Skodras, C. A. Christopoulos, T. Ebrahimi, "JPEG2000 : The Upcoming Still Image Compression Standard," IEEE Signal Processing Magazine, vol. 18, no. 5, pp 36 – 58, Septiembre 2001.

[4] G. Davis, A. Nosratinia, "Wavelet-based Image Coding: An Overview," Appl. Comp. Control, Signal & Circuits, Diciembre 1999, disponible online en http://www.geoffdavis.net/papers/accsc.pdf consultado en septiembre 2010.

[5] J. S. Walker, "Wavelet-Based Image Processing," University of Wisconsin, disponible online en http://www.uwec.edu/walkerjs/media/wbip.pdf consultado en febrero 2011.

[6] S. Subhasis, "Image Compression – from DCT to Wavelets: A Review," Crossroads, vol. 6, no. 3, pp. 12-21, Marzo 2000, disponible online en http://www.acm.org/crossroads/xrds6-3/sahaimgcoding.html consultado en noviembre de 2010.

[7] C.K. Chui, "An Introduction to Wavelets". vol. 1, Academic Press, San Diego 1992.

[8]H.G. Stark, "Wavelets and Signal Processing", Springer, Berlin, 2005.

[9] A. Mertins, "Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications". John Wiley & Sons Ltd, 1999.

[10] P.M. Saarbrucken, H.G.S. Kaiserslauternp, "Wavelets and Digital Image Processing", disponible online en http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=984CF8FEFEEFD86AF66B7741486073D2?doi=10.1.1.47.8581&rep=rep1&type=pdf consultado en noviembre 2010.

[11] I. Daubechies, "Ten Lectures on Wavelets", SIAM, Philadelphia, PA, Septiembre 1992.

[12] S. Grgic, M. Grgic, "Performance Analysis of Image Compression Using Wavelets," IEEE Transactions on Industrial Electronics, vol.48, no.3, pp. 682-695, Junio 2001.

[13] R. J. Colom, R Gadea, A. Sebastiá, M. Martinez, V. Herrero, V. Arnau, "Transformada Discreta Wavelet 2-D para Procesamiento de Vídeo en Tiempo Real," presentado en XII Jornadas de Paralelismo. Valencia, España, 2001, disponible online en http://www.uv.es/varnau/jor_pal_2001.pdf consultado en octubre 2010.

[14] M. Vetterli, C. Herley, "Wavelets and filter banks: Theory and design," IEEE Transactions on Signal Processing. vol. 40, no. 9, pp. 2207–2232, Septiembre 1992.

[15] S. Kouro, R. Masalem, "Tutorial Introductorio a la Teoria de Wavelet", Universidad Técnica Santa Maria, julio 2002, disponible online en http://www2.elo.utfsm.cl/~elo377/documentos/Wavelet.pdf consultado en noviembre 2011.

[16] M.L. Hilton, B.D. Jawerth, A. Sengupta, "Compressing still and moving images with wavelets," Multimedia Systems, vol. 2, no. 3, pp. 218–227, Abril 1994.

[17] J.S. Walker, "A Primer on Wavelets and Their Scientific Applications", Chapman & Hall/CRC Press, Boca Raton, New York, 1999.

[18] J.D. Villasenor, B. Belzer, J. Liao, "Wavelet filter evaluation for image compression," IEEE Transactions on image processing, vol. 4, no. 8, pp. 1053–1060, Agosto 1995.

[19] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, "Image Coding Using Wavelet Transform," IEEE Trans. Image Proc., vol. 1, pp. 205–220, Abril 1992.

[20] O. Rioul, "Regular wavelets: a discrete-time approach," IEEE Transactions on Signal Processing, vol. 41, no. 12, pp. 3572–3579, Diciembre 1993.

[21]M. Unser, "Approximation power of biorthogonal wavelet expansions," IEEE Transactions on Signal Processing, vol. 44, no. 3, pp. 519–527, Marzo 1996.

[22] G. Davis, A. Nosratinia, "Wavelet-based Image Coding: An Overview," Appl. Comp. Control, Signal & Circuits, Diciembre 1999, disponible online en http://www.geoffdavis.net/papers/accsc.pdf consultado en septiembre 2010.

[23] G. K. Kharate, V. H. Patil, N. L. Bhale, "Selection of Mother Wavelet for Image Compression on Basis of Nature of Image," Journal of Multimedia, vol. 2, no. 6, pp. 44-51, Noviembre 2007.

[24] Charles K. Chui, "Wavelets: a mathematical tool for signal processing", Ed. 2, SIAM, Philadelphia, 1997.

[25] Commission for Asia and the pacific, Asian-Pacific Remote Sensing and GIS Journal. vol. 14, United Nations Publications, Bangkok, 2002.

[26] C. Chen, Z. Yang, T. Wang, and L. Chen, "A programmable VLSI architecture for 2-D discrete wavelet transform," Procedente, IEEE International Symposium on Circuits and Systems, pp. 619-622, Mayo 2000.

[27] M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," IEEE Transactions on Circuits and Systems - II, vol. 42, no. 3, pp. 305-316, Mayo 1995.

[28] M. Vishwanath, "The Recursive Pyramid Algorithm for the discrete wavelet

transform," IEEE Transactions on Signal Processing, vol. 42, no. 3, pp. 673-676, Marzo 1994.

[29] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 7, pp. 674-693, Julio 1989.

[30] R. J. Colom, R. Gadea, A. Sebastiá, M. Martinez, F. Ballester, V. Herrero, "Implementación de la Transformada Wavelet Discreta 2D con filtros no separables," presentado en I Jornadas sobre Computación Reconfigurable y Aplicaciones. Alicante, España, 2001.

[31]M. Sheu, M. Shieh, S. Liu, "A VLSI Architecture Design With Lower Hardware Cost and Less Memory for separable 2-D Discrete Wavelet Transform," IEEE Circuits and Systems, Procedente de International Symposium on Circuits and Systems, vol. 5, pp. 457-459, Mayo 1998.

[32]C. Chakrabarti, C. Mumford, "Efficient Realizations of Encoders and Decoder Based on the 2-D Discrete Wavelet Transform," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, no. 3, pp. 289-298, Septiembre 1999.

[33] Cyclone II Device Handbook, Altera Corporation, Vol. 1, 2008, disponible online en http://www.altera.com/literature/lit-cyc2.jsp consultado en junio 2011.

[34] Timing I/O Analysis Reports, Altera Corporation, disponible online en http://quartushelp.altera.com/10.0/mergedProjects/report/rpt/rpt_file_tan_io_analysis.htm consultado en junio 2011.

[35] DE2 User Manual, Altera Corporation, disponible online en ftp://ftp.altera.com/up/pub/Webdocs consultado en junio 2010

[36] User Guide SOPC Builder, Altera Corporation, disponible online en http://www.altera.com/literature/lit-nio2.jsp#related_documentation consultado en junio 2011.

[37] Nios II Processor Reference Handbook, Altera Corporation, versión 11.0, 2011, disponible online en http://www.altera.com/literature/lit-nio2.jsp#related_documentation consultado en junio 2011.

[38] J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coeficients," IEEE Transactions on Signal Processing, vol. 41, no.12, pp. 3445-3462, Diciembre 1993.

[39] A. Said, W.A. Pearlman, "A new, fast, and eficient image codec based on set partitioning in hierarchical trees," IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 243-250, Junio 1996.

[40] J. Oliver, M.P. Malumbres, "Low complexity multiresolution image compression using wavelet lower trees," IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 11, pp. 1437 – 1444, Noviembre de 2006.

[41] D. Taubman, "High performance scalable image compression with EBCOT," IEEE Transactions on Image Processing, vol. 9, no. 7, pp. 1158 – 1170, Agosto de 2000.

[42] C.D. Creusere, "A new method of robust image compression Based on the Embedded Zerotree Wavelet Algorithm," IEEE Transactions on Image Processing, vol. 6, no. 10, pp. 1436 - 1442, Agosto de 2002.

[43] S.T. Hsiang, J.W. Woods, "Embedded Image Coding Using ZeroBlocks of Subband/Wavelet Coefficients and Context Modeling," IEEE Circuits and Systems, Procedente de International Symposium on Circuits and Systems, vol. 3, pp. 662 - 665, Agosto de 2002.

[44] Z. Zeng, I.G Cumming, "SAR image data compression Using a Tree-Structured Wavelet Transform," IEEE Transactions on Geoscience and Remote Sensing, vol. 39, no.3, pp. 546 - 552, Agosto de 2002.

[45] J.M. Shapiro, "An embedded wavelet hierarchical image coder," IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 4, pp. 657 - 660, Agosto de 2002.

[46] R.C. González, R.E. Woods, "Digital Image Processing", Prentice Hall, segunda edicion, New Jersey, 2002.

[47] M.D. Adams, "The JPEG-2000 Still Image Compression Standard", University of British Columbia, Canada, Septiembre 2001, disponible en http://www.ece.uvic.ca/~frodo/publications/jpeg2000.pdf consultado en febrero 2011.

[48] Grupo de Microelectrónica y Control de la Universidad de Antioquia, Grupo de Arquitecturas Digitales y Microelectrónica de la Universidad del Valle, "Apropiación de una tecnología para acompañantes móviles digitales", 2004.

[49] "Codificación y Compresión de Imágenes", disponible online en http://web.mac.com/ci3m/Site_3/Material_Adicional_files/CAP-7.PDF consultado en julio 2010.

[50] Z. Xiong, K. Ramchandran, M.T. Orchard, Y.Q. Zhang, "A comparative Study of DCT and Wavelet-Based Image Coding," IEEE Transactions On Circuits And Systems For Video Technology, vol. 9, no. 5, pp.692 – 695, Agosto 1999.

[51] L. Delgado, V.H. Rivera, "Uso de la Transformada Wavelets en el Procesamiento de Imágenes," Trabajo de Investigación desarrollado en la Universidad Nacional de San Agustin de Arequipa, Peru, 2008, disponible online en http://www.ciparequipa.org/TIDSPENI.pdf consultado en octubre 2010.

[52] K.Z. Bukhari, "Visual Data Transforms Comparison", Tesis de Maestría en Ingeniería Eléctrica, Delft University of Technology Netherlands, Agosto 2002.

[53] "Compresión de Vídeo Digital- Revisión de los Métodos y los Estándares a usar para la Transmisión y el Almacenamiento de Vídeo", Axis Communications, agosto 2004, disponible online en www.colegio28.comze.com/compresion_video_es.pdf consultado en noviembre 2010.

[54] C. Wootton, "A Practical Guide to Video and audio digital", Elsevier, Abril 2005.

[55] B.J. Kim, Z. Xiong, W.A. Pearlman, "Very Low Bit Rate Embedded Video Coding

whit 3D Set Partitioning in Hierarchical Trees (3D SPIHT)", Estudio desarrollado por el Centro para Procesamiento de Imágenes del Instituto Politécnico Rensselaer y la Universidad de Hawaii, noviembre 1997, disponible online en http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.3826&rep=rep1&type=pdf consultado en febrero 2011.

[56] G.V. Auwera, A. Munteanu, G. Lafruit, J. Cornelis, "Video Coding Based On Motion Estimation In The Wavelet Detail Images," IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 5, pp. 2801 - 2804, mayo 1998.

[57] Y. Bnjmohan, S.H. Mneney, "Low bit-rate video coding using fractal compression of wavelet subtrees," IEEE AFRICON Conference in Africa, vol. 1, no. 17, pp. 39 - 44, marzo 2005.

[58] M. Wei, H. Chao, "Rate Scalable Video Compression Based on Flexible Block Wavelet Coding Technique," IEEE DCC 2002 Data Compression Conference, pp. 478, agosto 2002.

[59] R. Adhami, "Video Compression Technique Using Wavelet Transform," IEEE Aerospace Applications Conference, vol. 4, no. 3, pp. 449 - 455, Agosto 2002.

[60] E.J. Balster, Y.F. Zheng, "Real-Time Video Rate Control Algorithm for a Wavelet-Based Compression Scheme," 44th IEEE 2001 Midwest Symposium on Circuits and Systems, vol. 1, pp. 492 - 496, Agosto 2002.

[61] C.I. Podilchuk, N.S. Jayant, N. Farvardin, "Three-dimensional subband coding of video," IEEE Transactions on Image Processing, vol. 4, no. 2, pp. 125 - 139, Agosto 2002.

[62] B. Felts, B. Pesquet-Popescu, "Efficient Context Modeling in Scalable 3D Wavelet-Based Video Compression," IEEE 2000 International Conference on Image Processing, vol. 1, pp. 1004 - 1007, Agosto 2002.

[63] A.S. Lewis, G. Knowles, "Video Compression Using 3D Wavelet Transforms," IEEE Electronics Letters, vol. 26, no. 6, pp. 396 - 398, Agosto 2002.

Anexo A.

Algoritmos Desarrollados en Matlab

A.1.-ANCHO DE BIT

Segmento de códigos para calcular el ancho de bit necesario dependiendo del nivel de transformación.

```
TB=9; %tamaño de bits de entrada
nivel=12;
%Inicializar los vectores
VminH=zeros(1,nivel+1); VmaxH=zeros(1,nivel+1); VminG=zeros(1,nivel+1); VmaxG=zeros(1,nivel+1);
%Calculo del mínimo y máximo de entrada
VminH(1)=-2^(TB-1); VminG(1)=VminH(1);
VmaxH(1)=2^(TB-1)-1;VmaxG(1)=VmaxH(1);
%Coeficientes de la Base
H=[-1,2,6,2,-1]; G=[-2,4,-2];
%Calculo de mínimo y máximo de los siguientes niveles
for I = 2:nivel+1,
  %Calculo para los coeficientes a la salida del FPasabajo
   for K= 1: length(H),
      if H(K)>0
         VminH(I)=VminH(I)+H(K)*VminH(I-1);
         VmaxH(I)=VmaxH(I)+H(K)*VmaxH(I-1);
      else
         VminH(I)=VminH(I)+H(K)*VmaxH(I-1);
         VmaxH(I)=VmaxH(I)+H(K)*VminH(I-1);
      end
   end
   %Calculo para los coeficientes a la salida del FPasaalto
   for K= 1: length(G),
      if G(K)>0
         VminG(I)=VminG(I)+G(K)*VminH(I-1);
         VmaxG(I)=VmaxG(I)+G(K)*VmaxH(I-1);
      else
         VminG(I)=VminG(I)+G(K)*VmaxH(I-1);
         VmaxG(I)=VmaxG(I)+G(K)*VminH(I-1);
      end
   end
```

```
   %Denormalizacion y Redondeo de valores
   if mod (I,2)==1
      VminH(I)=floor(VminH(I)/32);       VmaxH(I)=ceil(VmaxH(I)/32);
      VminG(I)=floor(VminG(I)/32);       VmaxG(I)=ceil(VmaxG(I)/32);
   end
end
%Calculo del numero de Bits
  if (abs(VminH))>VmaxH
      N=abs(VminH);
   else
      N=VmaxH;
   end
   NH=ceil(log2(2*N));
   if (abs(VminG))>VmaxG
      N=abs(VminG);
   else
      N=VmaxG;
   end
   NG=ceil(log2(2*N));
%Visualizacion de la Tabla
for I = 1:2:nivel+1,
   MATRIZ(I,1)=VminH(I);   MATRIZ(I,2)=VmaxH(I);
   MATRIZ(I,3)=VminG(I);   MATRIZ(I,4)=VmaxG(I);
   MATRIZ(I,5)=NH(I);       MATRIZ(I,6)=NG(I);
end
TABLA=['      VminH      VmaxH      VminG      VminG      bitsH      bitsG']
MATRIZ
```

A.2.-TRANSFORMADA WAVELET 2D

A.2.1- Coeficientes en Doble Punto Flotante

Función para calcular la transformada wavelet de una imagen usando la base biortogonal 2-2 con coeficientes en doble punto flotante.

```
function [FLL, FLH, FHL, FHH]=WcoefDoble()
I=imread( 'ImagenPrueba1.bmp' );
Ig=double(rgb2gray(I));
%Transformación primera etapa (filas)
[fil,col]=size(Ig);
[v,w]=VecTransBior22(col);     %Generación de los vectores de transformación para la primera etapa
v=v*sqrt(2)/8; w=w*sqrt(2)/8;
for j=1:fil %Transformacion
   [a(j,:),d(j,:)]=TransW_1D(col,v,w,Ig(j,:));
end
%Transformación segunda etapa (por columnas)
a=a';d=d';       %se calcula la transpuesta para procesar por columnas
[fil2,col2]=size(a);       r
[v2,w2]=VecTransBior22(col2);%Generación de los vectores de transformación para la segunda etapa
```

```
v2=v2*sqrt(2)/8; w2=w2*sqrt(2)/8;
for j=1:fil2
   [FLL(j,:),FLH(j,:)]=TransW_1D(col2,v2,w2,a(j,:));
   [FHL(j,:),FHH(j,:)]=TransW_1D(col2,v2,w2,d(j,:));
end
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH'; %se invierte el proceso de la transpuesta
return;
```

A.2.1- Coeficientes Enteros de 16 Bits

Función para calcular la transformada wavelet de una imagen usando la base biortogonal 2-2 con coeficientes enteros ajustados a 16 bits.

```
function [FLL, FLH, FHL, FHH,Ig]=WcoefEntero()
I=imread( 'lena.JPG' );
Ig=int16(rgb2gray(I));
%Transformación primera etapa (filas)
[fil,col]=size(Ig);
[v,w]=VecTransBior22(col);   %Generación de los vectores de transformación para la primera etapa
for j=1:fil %Transformación
   [a(j,:),d(j,:)]=TransW_1D(col,v,w,Ig(j,:));
 end
%Transformación segunda etapa (por columnas)
a=a';d=d';              %se calcula la transpuesta para procesar por columnas
[fil2,col2]=size(a);
[v2,w2]=VecTransBior22(col2);%Generación de los vectores de transformación para la segunda etapa
for j=1:fil2
   [FLL(j,:),FLH(j,:)]=TransW_1D(col2,v2,w2,a(j,:));
   [FHL(j,:),FHH(j,:)]=TransW_1D(col2,v2,w2,d(j,:));
end
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH'; %se invierte el proceso de la transpuesta
a=a';d=d';
factor=1/32;
FLL=int16(FLL*factor);FLH=int16(FLH*factor);FHL=int16(FHL*factor);FHH=int16(FHH*factor);
return;
```

A.3.-TRANSFORMADA WAVELET INVERSA 2D

A.3.1- Coeficientes en Doble Punto Flotante

Función para reconstruir una imagen a partir de su transformada wavelet usando la base biortogonal 2-2 con coeficientes en doble punto flotante.

```
function [IgR]=IWcoefDoble(FLL,FLH,FHL,FHH)
%Reconstrucción primera etapa
[fil2,col2]=size(FLL');
```

```
fil=col2*2;col=fil2*2;col2=fil;
[v2,w2]=VecInvTransBior22(col2);%Vectores de transformación para etapa de reconstrucción
v2=v2*sqrt(2)/8; w2=w2*sqrt(2)/8;
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH'; %se calculan las transpuestas para procesar por columnas
for j=1:fil2
   IgRa(j,:)=InvTransW_1D(col2,FLL(j,:),FLH(j,:),v2,w2);
   IgRd(j,:)=InvTransW_1D(col2,FHL(j,:),FHH(j,:),v2,w2);
end
IgRa=IgRa'; IgRd=IgRd'; %se invierte el proceso de la transpuesta
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH';
%Reconstrucción segunda etapa
[v,w]=VecInvTransBior22(col);
v=v*sqrt(2)/8; w=w*sqrt(2)/8;
for j=1:fil
   IgR(j,:)=InvTransW_1D(col,IgRa(j,:),IgRd(j,:),v,w);
end
return;
```

A.3.1- Coeficientes Enteros de 16 Bits

Función para reconstruir una imagen a partir de su transformada wavelet usando la base biortogonal 2-2 con coeficientes enteros ajustados a 16 bits.

```
function [IgR]=IWcoefEntero(FLL,FLH,FHL,FHH)
%Reconstrucción primera etapa
[fil2,col2]=size(FLL');
fil=col2*2;col=fil2*2;col2=fil;
[v2,w2]=VecInvTransBior22(col2);%Generación de los vectores de transformación para etapa de
reconstrucción
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH'; %se calculan las transpuestas para procesar por columnas
for j=1:fil2
   IgRa(j,:)=InvTransW_1D(col2,FLL(j,:),FLH(j,:),v2,w2);
   IgRd(j,:)=InvTransW_1D(col2,FHL(j,:),FHH(j,:),v2,w2);
end
IgRa=IgRa'; IgRd=IgRd'; %se invierte el proceso de la transpuesta
FLL=FLL';FLH=FLH';FHL=FHL';FHH=FHH';
%Reconstruccion segunda etapa
[v,w]=VecInvTransBior22(col);
for j=1:fil
   IgR(j,:)=InvTransW_1D(col,IgRa(j,:),IgRd(j,:),v,w);
end
IgR=int16(IgR*1/(32*1));
return;
```

A.4.-VECTORES DE TRANSFORMACIÓN

Función para generar los vectores de transformación de primer nivel usando la base

biortogonal 2-2. La función double( ) debe ser sustituida por int16 para las pruebas con coeficientes enteros.

```
function [v,w]=VecTransBior22(col)
%Coeficientes de la Wavelet Bior22
alfa1=-1;  alfa2=2;  alfa3=6;  alfa4=2;  alfa5=-1;  alfa6=0;
beta1=0;  beta2=0;  beta3=-2;  beta4=4;  beta5=-2;  beta6=0;
%Generación de la señal de escalamiento V y wavelet W de nivel 0
v(:,:)=zeros(col/2,col);
v(1,:)=[alfa1,alfa2,alfa3,alfa4,alfa5,alfa6,zeros(1,col-6)];
w(:,:)=zeros(col/2,col);
w(1,:)=[beta1,beta2,beta3,beta4,beta5,beta6,zeros(1,col-6)];
for i=2:col/2,
   v(i,:)=[v(i-1,col-1:col),v(i-1,1:col-2)];
   w(i,:)=[w(i-1,col-1:col),w(i-1,1:col-2)];
end
v=double(v);w=double(w);
return;
```

Función para generar los vectores de transformación inversa de primer nivel usando la base biortogonal 2-2. La función double() debe ser sustituida por int16 para las pruebas con coeficientes enteros.

```
function [v,w]=VecInvTransBior22(col)
%Coeficientes de la Wavelet Bior 2-2
alfa1=0;  alfa2=2;  alfa3=4;  alfa4=2;  alfa5=0;  alfa6=0;
beta1=0;  beta2=-1;  beta3=-2;  beta4=6;  beta5=-2;  beta6=-1;
%Generacion de la señal de escalamiento V y wavelet W de nivel 0
v(:,:)=zeros(col/2,col);
v(1,:)=[alfa1,alfa2,alfa3,alfa4,alfa5,alfa6,zeros(1,col-6)];
w(:,:)=zeros(col/2,col);
w(1,:)=[beta1,beta2,beta3,beta4,beta5,beta6,zeros(1,col-6)];
for i=2:col/2,
   v(i,:)=[v(i-1,col-1:col),v(i-1,1:col-2)];
   w(i,:)=[w(i-1,col-1:col),w(i-1,1:col-2)];
end
v=double(v);w=double(w);
return;
```

A.5.-TRANSFORMADA WAVELET 1D

Función para realizar la transformación wavelet 1D, usando los vectores de transformación, retorna las señales de fluctuación-detalle (a) y tendencia-aproximación (d). La función double() debe ser sustituida por int16 para las pruebas con coeficientes enteros.

```
function [a,d]=TransW_1D(col,v,w,Ig)
for i=1:col/2
    a(i)=sum(Ig.*v(i,:));
    d(i)=sum(Ig.*w(i,:));
end
a=double(a);d=double(d);
return;
```

A.6.-TRANSFORMADA WAVELET INVERSA 1D

Función para realizar la transformación inversa wavelet 1D, usando los vectores de transformación, retorna la imagen reconstruida. La función double() debe ser sustituida por int16 para las pruebas con coeficientes enteros.

```
function [IgR]=InvTransW_1D(col,a,d,v,w)
A=double(zeros(1,col));
D=double(zeros(1,col));
for i=1:col/2
   A=A+a(i)*v(i,:);
   D=D+d(i)*w(i,:);
end
IgR=A+D;
IgR=double(IgR);
return;
```

A.7.-ERROR CUADRÁTICO MEDIO

Función para realizar el cálculo del error cuadrático medio de dos imágenes.

```
function [R]=RSME(Io,Ir)
 [fil,col]=size(Io);
R=(Io-Ir).^2;
R=sum(sum(R))/(fil*col);
R=sqrt(R);
return;
```

A.8.-UNIDAD DE FILTROS

Función para validar la simulación de la Unidad de Filtros

```
function [a,d]=W1D( )
clear all; % limpiar el workspace
%Valores de la señal de entrada
F(1)=10;
F(2)=-100;
for k=2:1:24
   F(2*k-1)=F(2*k-3)+10;
   F(2*k)=F(2*k-2)+5;
end
[fil,col]=size(F);          %Se calcula el tamaño de la señal a procesar
[v,w]=VecTransBior22(col);   %Generación de los vectores de transformación
for j=1:fil %Transformación
   [a(j,:),d(j,:)]=TransW_1D(col,v,w,F);
end
return;
```

A.9.-ALGORITMO SPIHT

Programa para codificar una matriz mediante el algoritmo SPIHT.

```
function [salida]=SPIHTorg(I,N)
% I es una matriz que contiene los coeficientes de N niveles de transformación Wavelet
T=length(I)/(2^(N-1));%tamaño de la banda de nivel superior
% Paso de Inicialización
[LIP,lip,LIS,lis,TIPO,k,u]=Inicializacion(T,I);
ind=1;salida(1)=0; %Valores iniciales de las variables
while (k>=0)
   %Paso de Clasificación
   [salida,ind,LSP]=Clasificacion(I,T,u,lis,LIS,TIPO,LIP,lip,salida,ind);
   %Paso de Refinamiento
   for j=1:length(LSP)
      if I(LSP(j,1),LSP(j,2))>0
         I(LSP(j,1),LSP(j,2))=I(LSP(j,1),LSP(j,2))-u;
      else
         I(LSP(j,1),LSP(j,2))=I(LSP(j,1),LSP(j,2))+u;
      end
   end
   %Paso de Cuantificación
   k=k-1;    u=2^k;
end
return;

function [LIP,lip,LIS,lis,TIPO,k,u]=Inicializacion(T,I)
% Esta función realiza el paso de Inicialización para el algoritmo SPIHT
%Definir Umbral
C = max(max(abs(I)));  k= floor(log2(double(C))); u=2^k;
%colocar en LSP como lista vacía
LSP=0;
%colocar en LIP todos los coeficientes nivel superior
lip=1;
for fil=1:T
```

```
  for col=1:T
      LIP(lip,:)=[fil col];lip=lip+1;
  end;
end;
%colocar en LIS todas las raíces del mayor nivel exepto FLL
lis=1;
for fil=1:T
  for col=1:T
    if ((fil>T/2) | (col>T/2))
        LIS(lis,:)=[fil col];TIPO(lis)=(1);lis=lis+1;
    end;
  end;
end;
return;

function [salida,ind,LSP]=Clasificacion(I,T,u,lis,LIS,TIPO,LIP,lip,salida,ind)
% Esta función realiza el paso de Clasificación, ordena los coeficientes en las listas de acuerdo a la
significancia
%Codificar la Importancia de los coeficientes individuales de LIP
j=1;lsp=1;
while j<lip
  if abs(I(LIP(j,1),LIP(j,2)))<u
    %Coeficiente no significativo solo saca cero
    salida(ind)=00;ind=ind+1;
  else
    if I(LIP(j,1),LIP(j,2))>0
      %Coeficiente significativo positivo
      salida(ind)=10;ind=ind+1;
    else
      %Coeficiente significativo negativo
      salida(ind)=11;ind=ind+1;
    end
    %como es significativo debe pasarlo de LIP a LSP
    LSP(lsp,:)=LIP(j,:);lsp=lsp+1;
    for m=j:lip-2
      %eliminar el elemento y reorganizar la lista
      LIP(m,:)=LIP(m+1,:);
    end
    lip=lip-1;j=j-1;
  end
  j=j+1;
end
%Codificar la Importancia de los árboles de LIS
j=1;
while j<lis
   x=LIS(j,1);y=LIS(j,2);
   if TIPO(j)==1
     %Calcula la significancia
     S=SignificanciaArbol(x,y,I,TIPO(j),u);
     salida(ind)=S;ind=ind+1;
     if S==1
       %comprobar los cuatro hijos
       for m=0:3
```

```
            xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
            [LIP,lip,LSP,lsp,salida,ind]=SignificanciaPixel(xh,yh,I,u,LIP,lip,LSP,lsp,salida,ind);
         end;
         %Si tiene nietos Pasa al final de la lista y se ubica como TIPO2
         if (2*xh<=length(I) & 2*yh<=length(I) )
            LIS(lis,:)=[x y];TIPO(lis)=2;lis=lis+1;
         end;
         %eliminar el elemento de LIS y reorganizar la lista
         for m=j:lis-2
           LIS(m,:)=LIS(m+1,:);
           TIPO(m)=TIPO(m+1);
         end
        lis=lis-1;j=j-1;
      end
   else %Si es de tipo 2
      %Calcula la significancia
      S=SignificanciaArbol(x,y,I,TIPO(j),u);
      salida(ind)=S;ind=ind+1;
      if S==1
         %Ubicar los cuatro hijos en la lista LIS
         for m=0:3
            xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
            LIS(lis,:)=[xh yh];TIPO(lis)=1;lis=lis+1;
         end;
         %eliminar el elemento de LIS y reorganizar la lista
         for m=j:lis-2
            LIS(m,:)=LIS(m+1,:);
            TIPO(m)=TIPO(m+1);
         end
         lis=lis-1;j=j-1;
      end
   end
   j=j+1;
end
return;

function [LIP,lip,LSP,lsp,salida,ind]=SignificanciaPixel(x,y,I,u,LIP,lip,LSP,lsp,salida,ind)
% Esta función calcula la significancia de un coeficiente y  lo ubica en la lista correspondiente
if abs(I(x,y))<u
   %Coeficiente no significativo solo saca cero y se incluye en LIP
   salida(ind)=00;ind=ind+1;
   LIP(lip,:)=[x,y];lip=lip+1;
else
   LSP(lsp,:)=[x,y];lsp=lsp+1;
   if I(x,y)>0
      %Coeficiente significativo positivo
      salida(ind)=10;ind=ind+1;
   else
      %Coeficiente significativo negativo
      salida(ind)=11;ind=ind+1;
   end;
end;
return;
```

```
function [S]=SignificanciaArbol(x,y,I,TIPO,u)
% Esta función calcula la significancia de un árbol de tipo A o B
S=0;
%Comprobar Árbol
for m=0:3
    xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
    if (abs(I(xh,yh))>=u & TIPO==1)
       %Hijo significativo
       S=1;
       return;
    end;
    %Si tiene nietos evalúa
    if (2*xh<=length(I) & 2*yh<=length(I) )
       for n=0:3
          xn=2*xh-1+floor(n/2); yn=2*yh-1+mod(n,2);
          if abs(I(xn,yn))>=u
             %Nieto significativo
             S=1;
             return;
          end;
       end;
    end;
end;
return;
```

A.10.-ALGORITMO SPIHT MODIFICADO

Programa para codificar una matriz mediante el algoritmo SPIHT con las modificaciones propuestas.

```
function [salida]=SPIHTmod2(I,N)
% I es una matriz que contiene los coeficientes de N niveles de transformación Wavelet
T=length(I)/(2^(N-1));%tamaño de la banda de nivel superior
%Construir la matriz de signo
for fil=1:length(I)
  for col=1:length(I)
      if I(fil,col)<0
         %Coeficiente Negativo
         NS(fil,col)=1;
      else
         %Coeficiente Positivo
         NS(fil,col)=0;
      end
  end;
end;
% Paso de Inicialización
```

```
[LIS,lis,TIPO,k,u]=Inicializacion(T,I);
ind=1;salida(1)=0; %Valores iniciales de las variables
while (k>=0)
   %Paso de Clasificación
   [salida,ind,LSP]=Clasificacion(I,T,u,lis,LIS,TIPO,salida,ind);
   %Paso de Refinamiento
   for j=1:length(LSP)
      if I(LSP(j,1),LSP(j,2))>0
         I(LSP(j,1),LSP(j,2))=I(LSP(j,1),LSP(j,2))-u;
      else
         I(LSP(j,1),LSP(j,2))=I(LSP(j,1),LSP(j,2))+u;
      end
   end
   %Paso de Cuantificación
   k=k-1;    u=2^k;
end
return;

function [LIS,lis,TIPO,k,u]=Inicializacion(T,I)
% Esta función realiza el paso de Inicialización para el algoritmo SPIHT
%Definir Umbral
C = max(max(abs(I))); k= floor(log2(double(C))); u=2^k;
%colocar en LIS todas las raíces del mayor nivel excepto FLL
lis=1;
for fil=1:T
  for col=1:T
     if ((fil>T/2) | (col>T/2))
        LIS(lis,:)=[fil col];TIPO(lis)=(1);lis=lis+1;
     end;
  end;
end;
return;

function [salida,ind,LSP]=Clasificacion(I,T,u,lis,LIS,TIPO,salida,ind)
% Esta función realiza el paso de Clasificación, ordena los coeficientes en las listas de acuerdo a la
significancia
%Evaluar los coeficientes nivel superior
lsp=1;LSP(1,:)=[0 0];
for fil=1:T
  for col=1:T
     [LSP,lsp,salida,ind]=SignificanciaPixel(fil,col,I,u,LSP,lsp,salida,ind);
  end;
end;
%Codificar la Importancia de los árboles de LIS
j=1;
while j<lis
   x=LIS(j,1);y=LIS(j,2);
   if TIPO(j)==1
      %Calcula la significancia
      S=SignificanciaArbol(x,y,I,TIPO(j),u);
      salida(ind)=S;ind=ind+1;
      if S==1
         %comprobar los cuatro hijos
```

```
        for m=0:3
            xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
            [LSP,lsp,salida,ind]=SignificanciaPixel(xh,yh,I,u,LSP,lsp,salida,ind);
        end;
        %Si tiene nietos Pasa al final de la lista y se ubica como TIPO2
        if (2*xh<=length(I) & 2*yh<=length(I) )
            LIS(lis,:)=[x y];TIPO(lis)=2;lis=lis+1;
        end;
        %eliminar el elemento de LIS y reorganizar la lista
        for m=j:lis-2
          LIS(m,:)=LIS(m+1,:);
          TIPO(m)=TIPO(m+1);
        end
       lis=lis-1;j=j-1;
     end
   else %Si es de tipo 2
     %Calcula la significancia
     S=SignificanciaArbol(x,y,I,TIPO(j),u);
     salida(ind)=S;ind=ind+1;
     if S==1
        %Ubicar los cuatro hijos en la lista LIS
        for m=0:3
            xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
            LIS(lis,:)=[xh yh];TIPO(lis)=1;lis=lis+1;
        end;
        %eliminar el elemento de LIS y reorganizar la lista
        for m=j:lis-2
            LIS(m,:)=LIS(m+1,:);
            TIPO(m)=TIPO(m+1);
        end
        lis=lis-1;j=j-1;
     end
   end
   j=j+1;
end
return;

function [LSP,lsp,salida,ind]=SignificanciaPixel(x,y,I,u,LSP,lsp,salida,ind)
% Esta función calcula la significancia de un coeficiente y  lo ubica en la lista correspondiente
if abs(I(x,y))<u
   %Coeficiente no significativo solo saca cero
   salida(ind)=00;ind=ind+1;
else
   LSP(lsp,:)=[x,y];lsp=lsp+1;
   salida(ind)=1;ind=ind+1;
end;
return;

function [S]=SignificanciaArbol(x,y,I,TIPO,u)
% Esta función calcula la significancia de un árbol de tipo A o B
S=0;
%Comprobar Árbol
for m=0:3
```

```
    xh=2*x-1+floor(m/2); yh=2*y-1+mod(m,2);
    if (abs(I(xh,yh))>=u & TIPO==1)
       %Hijo significativo
       S=1;
       return;
    end;
    %Si tiene nietos evalúa
    if (2*xh<=length(I) & 2*yh<=length(I) )
       for n=0:3
          xn=2*xh-1+floor(n/2); yn=2*yh-1+mod(n,2);
          if abs(I(xn,yn))>=u
             %Nieto significativo
             S=1;
             return;
          end;
       end;
    end;
end;
return;
```

Anexo B.

Módulos Desarrollados en VHDL

B.1.-REGISTRO ACUMULADOR

Registro utilizado como acumulador en la unidad de filtros, el dato de entrada pasa a las salidas en los flancos de subida de la señal de reloj, tiene dos señales de salida (Q1, Q2), una de las cuales (Q1) tiene asociada un borrado asíncrono activo alto que se utiliza para borrar el acumulador cuando se inicia el cálculo de un nuevo coeficiente. La otra señal (Q2) permanece con el último valor para garantizar el almacenamiento del dato previo. La tabla B.1 muestra los recursos y requerimientos de tiempo en la implementación de este modulo.

El código utilizado para modelar este registro es el siguiente:

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Registro is
generic (   N2      : integer :=15);--Ancho del bus de datos
port (      CLK     : in std_logic;
            Limpia: in std_logic;
            D       : in std_logic_vector(N2 downto 0);
            Q1,Q2 : out std_logic_vector(N2 downto 0) );
end Registro;

architecture Behavioral of Registro is
signal aux: std_logic_vector(N2 downto 0);
begin
aux<="0000000000000000" when (Limpia = '1') else
    D when ( CLK'event and CLK ='1');
Q1<= aux after 2 ns;
```

```
Q2 <=  D when ( CLK'event and CLK ='1');
end Behavioral;
```

Tabla B. 1 Recursos y Requerimientos de Tiempo del Modulo Registro

Tipo	**Recursos Utilizados**
Elementos Lógicos	32/33216 (<1%)
Registros	32
Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	0/70 (0%)
PLL	0/4 (0%)
Peor Caso Tsu	3,418 ns
Peor Caso Tco	6,816 ns
Peor Caso Th	0,837 ns

Donde tsu (timing requirements for setup) es el requerimiento de tiempo de configuración o tiempo de estabilización de la entrada que alimenta el reloj, tco (timing requirements for clock-to-output) es el requerimiento de tiempo de estabilización entre el reloj y la salida, th (timing requirements for hold) es el requerimiento de tiempo entre una entrada y el registro destino [34].

B.2.-REGISTRO DE COEFICIENTES

Este registro permite cargar los coeficientes del filtro usando una señal (Load) asíncrona activa alta, este registro rota los coeficientes dos posiciones con el flanco de subida del reloj. La tabla B.2 muestra los recursos y requerimientos de tiempo en la implementación de este modulo.

El código utilizado para modelar este registro es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RegCoeficientes is
generic (     N1: integer :=6;          --Coeficientes del filtro
              N3: integer :=3;          --Ancho de bit de los Coeficientes
              COE1: std_logic_vector(3 downto 0)  :="1111"; --Coeficientes
              COE2: std_logic_vector(3 downto 0) :="0010";
              COE3: std_logic_vector(3 downto 0) :="0110";
              COE4: std_logic_vector(3 downto 0) :="0010";
              COE5: std_logic_vector(3 downto 0) :="1111";
              COE6: std_logic_vector(3 downto 0) :="0000");
port (        CLK,Carga              : in std_logic;
              C1,C2,C3,C4,C5,C6      : out std_logic_vector(N3 downto 0) );
end RegCoeficientes;

architecture Behavioral of RegCoeficientes is
signal Q1,Q2,Q3,Q4,Q5,Q6: std_logic_vector(N3 downto 0);
begin
Q1 <= COE1 when (Carga = '1') else     Q3 when ( CLK'event and CLK ='1');
Q2 <= COE2 when (Carga = '1') else     Q4 when ( CLK'event and CLK ='1');
Q3 <= COE3 when (Carga = '1') else     Q5 when ( CLK'event and CLK ='1');
Q4 <= COE4 when (Carga = '1') else     Q6 when ( CLK'event and CLK ='1');
Q5 <= COE5 when (Carga = '1') else     Q1 when ( CLK'event and CLK ='1');
Q6 <= COE6 when (Carga = '1') else     Q2 when ( CLK'event and CLK ='1');
C1<=Q1 after 2 ns;
C2<=Q2 after 2 ns;
C3<=Q3 after 2 ns;
C4<=Q4 after 2 ns;
C5<=Q5 after 2 ns;
C6<=Q6 after 2 ns;
end Behavioral;
```

Tabla B. 2 Recursos y Requerimientos de Tiempo del Modulo Registro de Coeficientes

Tipo	**Recursos Utilizados**
Elementos Lógicos	15/33216 (<1%)
Registros	15
Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	0/70 (0%)
PLL	0/4 (0%)
Peor Caso Tco	8.885 ns

B.3.-SUMADOR

Este modulo permite sumar tres datos. La tabla B.3 muestra los recursos y requerimientos de tiempo en la implementación de este modulo utilizando la megafunción de altera parallel_add y utilizando el siguiente segmento de código:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Sumador is
generic (     N2    : integer :=15); --Ancho del bus de datos
port (        A,B,C: in std_logic_vector(N2 downto 0);
              F: out std_logic_vector(N2 downto 0)      );
end Sumador;

architecture Behavioral of Sumador is
begin
F<=A+B+C;
End Behavioral;
```

Tabla B. 3 Recursos y Requerimientos de Tiempo del Modulo Sumador

MEGAFUNCION		**VHDL**	
Tipo	**Recursos Utilizados**	**Tipo**	**Recursos Utilizados**
Elementos Lógicos	32/33216 (<1%)	Elementos Lógicos	32/33216 (<1%)
Registros	0	Registros	0
Bits de Memoria	0/483840 (0%)	Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	0/70 (0%)	Multiplicadores	0/70 (0%)
PLL	0/4 (0%)	PLL	0/4 (0%)
Peor Caso Tpd	13,055 ns	Peor Caso Tpd	13,632 ns

Donde tpd (timing requirements for point-to-point delay) es el retardo punto a punto entre la fuente y el nodo destino [34].

En la tabla B.3 se puede observar que el sumador implementado con la megafunción parallel_add requiere un tiempo de estabilización menor en 0,577ns que el diseño en

VHDL y que los dos sumadores requieren la misma cantidad de elementos lógicos, por lo tanto en la unidad de filtros se utiliza el sumador de la megafunción.

B.4.-MULTIPLEXOR

Este módulo permite seleccionar un dato de salida entre tres datos de entrada. La tabla B.4 muestra los recursos y requerimientos de tiempo en la implementación de este módulo utilizando la megafunción de altera lpm_mux y utilizando el siguiente segmento de código:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Multiplexor is
generic (     N2    : integer :=15); --Ancho del bus de datos
port (        D1,D2,D3                : in std_logic_vector(N2 downto 0);
              C1,C0                   : in std_logic;
              F               : out std_logic_vector(N2 downto 0)          );
end Multiplexor;

architecture Behavioral of Multiplexor is
begin
F <=    D1 when ( C1='0' and C0 ='1') else
                  D2 when ( C1='1' and C0 ='0') else
                  D3 when ( C1='1' and C0 ='1');
end Behavioral;
```

Tabla B. 4 Recursos y Requerimientos de Tiempo del Modulo Multiplexor

MEGAFUNCIÓN		**VHDL**	
Tipo	**Recursos Utilizados**	**Tipo**	**Recursos Utilizados**
Elementos Lógicos	32/33216 (<1%)	Elementos Lógicos	49/33216 (<1%)
Registros	0	Registros	0
Bits de Memoria	0/483840 (0%)	Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	0/70 (0%)	Multiplicadores	0/70 (0%)

PLL	0/4 (0%)	PLL	0/4 (0%)
Peor Caso Tpd	12,576 ns	Peor Caso Tco	9,454 ns

En la tabla B.4 se puede observar que el multiplexor implementado con la megafunción lpm_mux requiere 17 elementos lógicos menos que el diseño en VHDL y un tiempo de estabilización mayor; en la unidad de filtros se utilizara la megafunción dado que el tiempo de estabilización del sumador es superior al tiempo de ambos multiplexores con lo cual no es posible sacar provecho de esta característica y es de mas utilidad el modulo con menos elementos lógicos.

B.4.-MULTIPLICADOR

Para la implementación de los multiplicadores se utilizo la megafunción de altera lpm_mul que permite utilizar los multiplicadores embebidos optimizando el uso de la FPGA. La tabla B.5 muestra los recursos y requerimientos de tiempo en la implementación de este módulo.

Tabla B. 5 Recursos y Requerimientos de Tiempo del Modulo Multiplicador

Tipo	**Recursos Utilizados**
Elementos Lógicos	0/33216 (0%)
Registros	0
Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	2/70 (3%)
PLL	0/4 (0%)
Peor Caso Tpd	15,227 ns

B.4.-UNIDAD DE FILTROS

En la tabla B.6 se muestra los recursos y requerimientos de tiempo en la implementación de esta unidad. La conexión interna de los elementos que la componen siguiendo el esquema de la figura 4.11 se describe en el siguiente módulo VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UF is
generic (       N1: integer :=6;--Coeficientes del filtro
                N2: integer :=15;--Ancho del bus de datos
                N3: integer :=3);--Ancho de bit de los Coeficientes
port (          CLK,Carga: in std_logic;
                Limpiar  : in std_logic_vector(3 downto 1);
                Sel : in std_logic_vector(1 downto 0);
                Impar, Par: in std_logic_vector(N2 downto 0);
                SalidaH,SalidaG  : out std_logic_vector(N2 downto 0) );
end UF;


architecture Behavioral of UF is

component RegCoeficientesH
generic (       N1: integer :=6;--Coeficientes del filtro
                N3: integer :=3);--Ancho de bit de los Coeficientes
port (          CLK,Carga: in std_logic;
                C1,C2,C3,C4,C5,C6: out std_logic_vector(N3 downto 0) );
end component;

component RegCoeficientesG
generic (       N1: integer :=6;--Coeficientes del filtro
                N3: integer :=3);--Ancho de bit de los Coeficientes
port (          CLK,Carga : in std_logic;
                C1,C2,C3,C4,C5,C6: out std_logic_vector(N3 downto 0) );
end component;

component lpm_mult0
PORT    (       dataa     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                datab     : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                result    : OUT STD_LOGIC_VECTOR (19 DOWNTO 0));
end component;

component parallel_add0
PORT    (       data0x    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                data1x    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                data2x    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                result    : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
```

```
end component;

component Registro
generic (       N2      : integer :=15);--Ancho del bus de datos
port (          CLK     : in std_logic;
                Limpia  : in std_logic;
                D       : in std_logic_vector(N2 downto 0);
                Q1,Q2   : out std_logic_vector(N2 downto 0) );
end component;

component lpm_mux0
PORT    (       data0x  : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                data1x  : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                data2x  : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                sel     : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                result  : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
end component;

--Definir tipos de datos para la conexión
type Arreglo1 is array (N1 downto 1) of std_logic_vector( N3 downto 0);
type Arreglo2 is array (N1 downto 1) of std_logic_vector( N2 downto 0);
type Arreglo3 is array (N1/2 downto 1) of std_logic_vector( N2 downto 0);

--Señales de Conexión Interna
signal Conex1H,Conex1G : Arreglo1;
signal Conex2H,Conex3H,Conex2G,Conex3G : Arreglo2;
signal Conex4H,Conex5H, Conex4G,Conex5G : Arreglo3;

begin
--UNIDAD DE FILTRADO PARTE BAJA
--Conexión del Registro de Coeficientes
ConexRegCoefH : RegCoeficientesH port map (CLK=>CLK, Carga=>Carga, C1=>Conex1H(1),
C2=>Conex1H(2), C3=>Conex1H(5), C4=>Conex1H(6),C5=>Conex1H(3),C6=>Conex1H(4));
--Conexión de Multiplicadores
ConexMultH : for i in 1 to N1/2 generate
     Mult1H:lpm_mult0 port map (dataa=>Conex1H(2*i-1), datab=>Impar,result(15 downto
0)=>Conex2H(2*i-1));
     Mult2H: lpm_mult0 port map (dataa=>Conex1H(2*i), datab=>Par,result(15 downto
0)=>Conex2H(2*i));
end generate ConexMultH;
--Conexión de Sumadores
ConexSumH : for i in 1 to N1/2 generate
          SumH: parallel_add0 port map (data0x=>Conex2H(2*i-1), data1x=>Conex2H(2*i),
data2x=>Conex5H(i),
          result=>Conex3H(i));
end generate ConexSumH;
--Conexión de Registros
ConexRegH : for i in 1 to N1/2 generate
          RegH: Registro port map (CLK=>CLK, Limpia=>Limpiar(i), D=>Conex3H(i), Q1=>Conex5H(i),
          Q2=>Conex4H(i));
end generate ConexRegH;
--Conexión del Multiplexor
ConexMuxH: lpm_mux0 port map (data0x=>Conex4H(1),data1x=>Conex4H(2),data2x=>Conex4H(3),
```

```
sel=>Sel, result=>SalidaH);

--UNIDAD DE FILTRADO PARTE ALTA
--Conexión del Registro de Coeficientes
ConexRegCoefG : RegCoeficientesG port map (CLK=>CLK, Carga=>Carga, C1=>Conex1G(1), C2=>Conex1G(2), C3=>Conex1G(5), C4=>Conex1G(6), C5=>Conex1G(3),C6=>Conex1G(4));
--Conexión de Multiplicadores
ConexMultG : for i in 1 to N1/2 generate
    Mult1G:lpm_mult0 port map (dataa=>Conex1G(2*i-1), datab=>Impar,result(15 downto 0)=>Conex2G(2*i-1));
    Mult2G: lpm_mult0 port map (dataa=>Conex1G(2*i), datab=>Par,result(15 downto 0)=>Conex2G(2*i));
end generate ConexMultG;
--Conexión de Sumadores
ConexSumG : for i in 1 to N1/2 generate
        SumG: parallel_add0 port map (data0x=>Conex2G(2*i-1), data1x=>Conex2G(2*i), data2x=>Conex5G(i),
        result=>Conex3G(i));
end generate ConexSumG;
--Conexión de Registros
ConexRegG : for i in 1 to N1/2 generate
        RegG: Registro port map (CLK=>CLK, Limpia=>Limpiar(i), D=>Conex3G(i), Q1=>Conex5G(i),
        Q2=>Conex4G(i));
end generate ConexRegG;
--Conexión del Multiplexor
ConexMuxG: lpm_mux0 port map (data0x=>Conex4G(1),data1x=>Conex4G(2),data2x=>Conex4G(3), sel=>Sel, result=>SalidaG);

end Behavioral;
```

Tabla B. 6 Recursos y Requerimientos de Tiempo de la Unidad de Filtros

Tipo	**Recursos Utilizados**
Elementos Lógicos	292/33216 (<1%)
Registros	219
Bits de Memoria	0/483840 (0%)
Multiplicadores Embebidos	24/70 (34%)
PLL	0/4 (0%)
Peor Caso Tsu	12.760 ns
Peor Caso Tco	10.286 ns
Peor Caso Tpd	12.423 ns
Peor Caso Th	7.119 ns

B.5.- MEMORIA

El código utilizado para modelar unidades de memoria de diferentes tamaños es el siguiente:

```
Entity BancoDeMemoria is
Port (
        DirW    : in integer range 2047 downto 0;   --Dirección Escritura
        W       : in std_logic;                     --Habilitación de Escritura
        D       : in std_logic_vector(7 downto 0);  --Dato De Entrada
        QA      : out std_logic_vector(7 downto 0); --Dato De Salida
       CLK     : in std_logic   );
End BancoDeMemoria;

Architecture Comportamental of BancoDeMemoria is
Type ArregloBanco is array (2047 downto 0) of std_logic_vector(7 downto 0);
Signal Banco: ArregloBanco;
Begin
Process(CLK,W,DirW,Banco)
Begin
                QA<=Banco(DirW);
                If (W='1' and (CLK='0' and CLK'event)) then
                        Banco(DirW)<=D;
                End if;
End process;
End Comportamental;
```

Anexo C.

Algoritmos Desarrollados en C++

C.1.-PROGRAMA DE LA UNIDAD DE CONTROL

El programa desarrollado en Altera IDE para la unidad de control basada en el sistema Nios II es el siguiente:

```
/* Librerias */
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "system.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "sys/alt_flash.h"
#include "altera_avalon_pio_regs.h"

/* Direcciones Base Tomadas del SOPC */
#define SDRAM_BASE_ADDRESS 0x00000000
#define SYNC1_BASE_ADDRESS 0x008818B0
#define SYNC2_BASE_ADDRESS 0x008818C0
#define SYNC3_BASE_ADDRESS 0x008818D0
#define LOAD1_BASE_ADDRESS 0x008818E0
#define LOAD2_BASE_ADDRESS 0x008818F0
#define LOAD3_BASE_ADDRESS 0x00881900
#define CLEAR1_BASE_ADDRESS 0x00881910
#define CLEAR2_BASE_ADDRESS 0x00881920
#define CLEAR3_BASE_ADDRESS 0x00881930
#define SEL1_BASE_ADDRESS 0x00881940
#define SEL2_BASE_ADDRESS 0x00881950
#define SEL3_BASE_ADDRESS 0x00881960
#define IMPAR1_BASE_ADDRESS 0x00881970
#define IMPAR2_BASE_ADDRESS 0x00881980
```

```
#define IMPAR3_BASE_ADDRESS 0x00881990
#define PAR1_BASE_ADDRESS 0x008819A0
#define PAR2_BASE_ADDRESS 0x008819B0
#define PAR3_BASE_ADDRESS 0x008819C0
#define DATOH1_BASE_ADDRESS 0x008819D0
#define DATOH2_BASE_ADDRESS 0x008819E0
#define DATOH3_BASE_ADDRESS 0x008819F0
#define DATOG1_BASE_ADDRESS 0x00881A00
#define DATOG2_BASE_ADDRESS 0x00881A10
#define DATOG3_BASE_ADDRESS 0x00881A20
#define IMAGEN_BASE_ADDRESS 0
#define FL1_BASE_ADDRESS 262144
#define FH1_BASE_ADDRESS 393216
#define FLL1_BASE_ADDRESS 524288
#define FLH1_BASE_ADDRESS 589824
#define FHL1_BASE_ADDRESS 655360
#define FHH1_BASE_ADDRESS 720896
#define FL1T_BASE_ADDRESS 1835008
#define Filas 16
#define Columnas 16

/*Función para cargar la Imagen de Prueba*/
void CargarImagenPrueba(void)
{
   int IndiceFilas=0, IndiceColumnas=0,i=0;
   alt_16 F[Filas][Columnas]={{163, 162, 161, 160, 163, 157, 163, 162, 165, 161, 163, 161, 154, 165, 159,
154},
   {162, 162, 162, 161, 163, 156, 163, 162, 165, 161, 162, 160, 154, 165, 159, 154},
   {162, 162, 163, 161, 163, 157, 163, 162, 166, 161, 161, 160, 155, 164, 160, 155},
   {162, 163, 163, 161, 162, 157, 163, 162, 166, 162, 160, 161, 155, 163, 161, 155},
   {164, 164, 162, 159, 161, 157, 164, 162, 164, 162, 158, 161, 154, 160, 160, 154},
   {163, 163, 161, 157, 161, 157, 163, 159, 161, 161, 156, 161, 153, 156, 158, 152},
   {161, 161, 159, 157, 161, 158, 161, 155, 159, 161, 157, 162, 152, 153, 156, 151},
   {159, 159, 158, 156, 162, 158, 160, 152, 158, 162, 158, 163, 153, 153, 156, 151},
   {155, 156, 157, 157, 158, 159, 159, 158, 168, 157, 159, 159, 162, 154, 157, 156},
   {155, 156, 157, 158, 158, 157, 155, 154, 161, 155, 157, 156, 158, 154, 157, 154},
   {156, 156, 157, 159, 158, 154, 151, 151, 158, 156, 159, 156, 156, 155, 158, 155},
   {157, 156, 157, 159, 158, 154, 151, 151, 160, 158, 159, 158, 156, 155, 157, 157},
   {157, 156, 157, 158, 158, 155, 154, 155, 160, 156, 156, 159, 155, 153, 155, 159},
   {158, 157, 157, 157, 157, 157, 157, 157, 159, 154, 153, 160, 155, 154, 155, 160},
   {157, 159, 159, 156, 156, 158, 158, 156, 158, 155, 154, 161, 155, 157, 156, 158},
   {157, 160, 160, 156, 155, 158, 157, 153, 156, 157, 156, 161, 154, 158, 155, 153}};
   for( i = IMAGEN_BASE_ADDRESS; ((IndiceColumnas<Columnas)&&(IndiceFilas<Filas)); i++ )
   {
      IOWR(SDRAM_BASE_ADDRESS, i,F[IndiceFilas][IndiceColumnas]);
      if ((IndiceColumnas==(Columnas-1))&&(IndiceFilas<(Filas-1)))
          {
            IndiceColumnas=0;
            IndiceFilas++;
          }
      else
          IndiceColumnas++;
   }
```

```
   return;
}

/*Función para realizar la transformación Op=1 filas  Op=2 Columnas*/
int DWT1D( int Op,int C ,int F, int i, int *m, int *n ,int *sync,int *load,int *sel,int *clear, alt_16 *impar, alt_16 *par, alt_16 *datoh, alt_16 *datog)
{
   int aux,salto,salto2,Indicador;
   if (Op==1)
   {     salto=1;     salto2=2;     Indicador=C;   }
   else
   {     salto=C;     salto2=2*C;     Indicador=F;    }

   //inicialización de señales de control
   *(sync)=1;
   //Cargar Datos
   *(impar)=IORD(SDRAM_BASE_ADDRESS, i);
   *(par)=IORD(SDRAM_BASE_ADDRESS, i+ salto);
   *(load)=1;
   *(sel)=0;
   *(clear)=7;
   usleep(4); /* Wait 4 us */

   *(sync)=0;
   *(load)=0;
   *(clear)=6;
   usleep(4);

   *(sync)=1;
    i =i+salto2;
   *(impar)=IORD(SDRAM_BASE_ADDRESS, i);
   *(par)=IORD(SDRAM_BASE_ADDRESS, i+salto);
   usleep(4);

   *(sync)=0;
   *(clear)=4;
   usleep(4);

   *(sync)=1;
      i =i+salto2;
   *(impar)=IORD(SDRAM_BASE_ADDRESS, i);
   *(par)=IORD(SDRAM_BASE_ADDRESS, i+salto);
   usleep(4);

   *(sync)=0;
   *(clear)=0;
   usleep(4);

   aux = 3;
   while(1)
   {
      *(sync)=1;
      aux++;
```

```
i =i+salto2;
if  (aux>Indicador/2)
{ *(sel)=0;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  *(sel)=1;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  *(sel)=2;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  break;
}
*(impar)=IORD(SDRAM_BASE_ADDRESS, i);
*(par)=IORD(SDRAM_BASE_ADDRESS, i+salto);
*(sel)=0;
*(clear)=1;
 usleep(4); /* Wait 4 us */

*(sync)=0;
*(load)=0;
*(clear)=0;
IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
*m =*m+salto;
*n =*n+salto;
usleep(4);

*(sync)=1;
aux++;
i =i+salto2;
if  (aux>Indicador/2)
{ *(sel)=1;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  *(sel)=2;
  usleep(4); /* Wait 4 us */
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
```

```
  *(sel)=0;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  break;
 }
*(impar)=IORD(SDRAM_BASE_ADDRESS, i);
*(par)=IORD(SDRAM_BASE_ADDRESS, i+salto);
*(sel)=1;
*(clear)=2;
usleep(4);

*(sync)=0;
*(clear)=0;
IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
*m =*m+salto;
*n =*n+salto;
usleep(4);

*(sync)=1;
aux++;
i =i+salto2;
if  (aux>Indicador/2)
{ *(sel)=2;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  *(sel)=0;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  *(sel)=1;
  usleep(4);
  IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
  IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
  *m =*m+salto;
  *n =*n+salto;
  break;
 }
*(impar)=IORD(SDRAM_BASE_ADDRESS, i);
*(par)=IORD(SDRAM_BASE_ADDRESS, i+salto);
*(sel)=2;
*(clear)=4;
usleep(4);

*(sync)=0;
```

```
      *(clear)=0;
      IOWR(SDRAM_BASE_ADDRESS, *m,*(datoh));
      IOWR(SDRAM_BASE_ADDRESS, *n,*(datog));
      *m =*m+salto;
      *n =*n+salto;
      usleep(4);
    }

 if (Op==2)
 {    *m=*m-C*F/2+1;    *n=*n-C*F/2+1;  }

 return i;
}


int main(void)
{
   //Define las variables
   int * sync1 = (int *)SYNC1_BASE_ADDRESS;
   int * sync2 = (int *)SYNC2_BASE_ADDRESS;
   int * sync3 = (int *)SYNC3_BASE_ADDRESS;
   int * load1 = (int *)LOAD1_BASE_ADDRESS;
   int * load2 = (int *)LOAD2_BASE_ADDRESS;
   int * load3 = (int *)LOAD3_BASE_ADDRESS;
   int * clear1 = (int *)CLEAR1_BASE_ADDRESS;
   int * clear2 = (int *)CLEAR2_BASE_ADDRESS;
   int * clear3 = (int *)CLEAR3_BASE_ADDRESS;
   int * sel1 = (int *)SEL1_BASE_ADDRESS;
   int * sel2 = (int *)SEL2_BASE_ADDRESS;
   int * sel3 = (int *)SEL3_BASE_ADDRESS;
   alt_16 * impar1 = (alt_16 *)IMPAR1_BASE_ADDRESS;
   alt_16 * impar2 = (alt_16 *)IMPAR2_BASE_ADDRESS;
   alt_16 * impar3 = (alt_16 *)IMPAR3_BASE_ADDRESS;
   alt_16 * par1 = (alt_16 *)PAR1_BASE_ADDRESS;
   alt_16 * par2 = (alt_16 *)PAR2_BASE_ADDRESS;
   alt_16 * par3 = (alt_16 *)PAR3_BASE_ADDRESS;
   alt_16 * datoh1 = (alt_16 *)DATOH1_BASE_ADDRESS;
   alt_16 * datoh2 = (alt_16 *)DATOH2_BASE_ADDRESS;
   alt_16 * datoh3 = (alt_16 *)DATOH3_BASE_ADDRESS;
   alt_16 * datog1 = (alt_16 *)DATOG1_BASE_ADDRESS;
   alt_16 * datog2 = (alt_16 *)DATOG2_BASE_ADDRESS;
   alt_16 * datog3 = (alt_16 *)DATOG3_BASE_ADDRESS;
   int i,i2,i3;
   int aux1=0;
   int m,n,m2,n2;

   CargarImagenPrueba();

 //Transformación por filas
 m=FL1_BASE_ADDRESS;
 n=FH1_BASE_ADDRESS;
 i = IMAGEN_BASE_ADDRESS;
 for( aux1=0; aux1<Filas; aux1++ )
```

```
   {
    i= DWT1D( 1,Columnas ,Filas,i,&m,&n ,(int *)sync1,(int *)load1,(int *)sel1,(int *)clear1, (alt_16 *)impar1, (alt_16 *)par1, (alt_16 *)datoh1, (alt_16 *)datog1);
   };

  //Transformación por Columnas
  m=FLL1_BASE_ADDRESS;
  n=FLH1_BASE_ADDRESS;
  i = FL1_BASE_ADDRESS;
  m2=FHL1_BASE_ADDRESS;
  n2=FHH1_BASE_ADDRESS;
  i2 = FH1_BASE_ADDRESS;
  for( aux1=0; aux1<Columnas/2; aux1++ )
   {
     i3= DWT1D( 2,Columnas/2 ,Filas,i,&m,&n ,(int *)sync2,(int *)load2,(int *)sel2,(int *)clear2, (alt_16 *)impar2, (alt_16 *)par2, (alt_16 *)datoh2, (alt_16 *)datog2);
     i3= DWT1D( 2,Columnas/2 ,Filas,i2,&m2,&n2 ,(int *)sync3,(int *)load3,(int *)sel3,(int *)clear3, (alt_16 *)impar3, (alt_16 *)par3, (alt_16 *)datoh3, (alt_16 *)datog3);
     i++;
     i2++;
   };

   return 0;
}
```